ENCICLOPEDIA PRACTICA DE LA

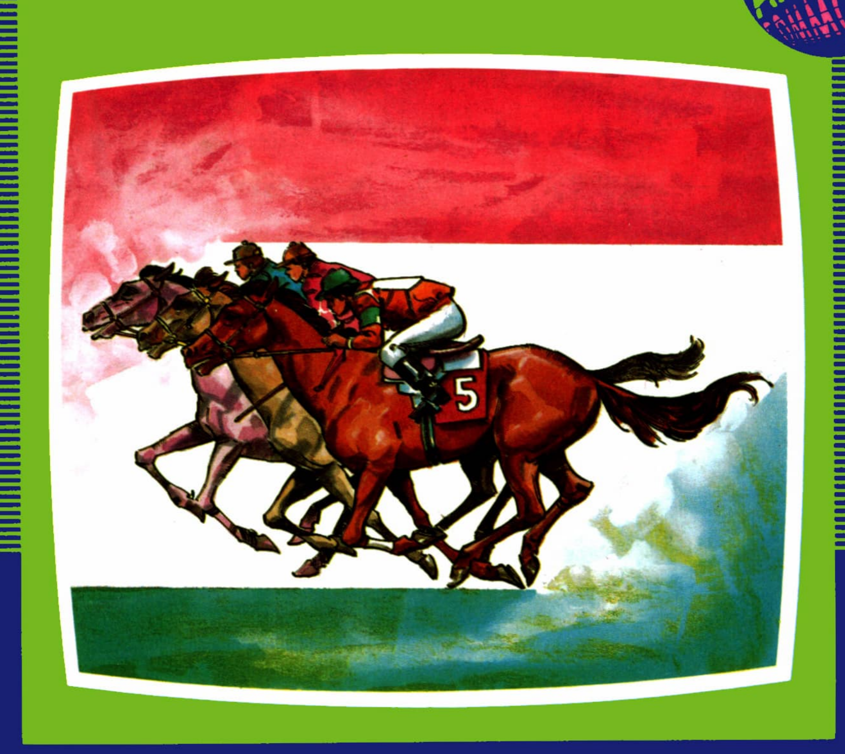
INFORMATICA APLICADA

6

Gráficos animados con el ordenador

Aula de Informática

SINCLAIR
AMSTRAD
COMMODORE
IBM



EDICIONES SIGLO CULTURAL

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

6

Gráficos animados con el ordenador

EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática
JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño y maquetación:
BRAVO-LOFISH.

Dibujos:
JOSE OCHOA Y ANTONIO PERERA.

Tomo 6. **Gráficos animados con el ordenador**
ESTHER MALDONADO, Diplomada en Arquitectura

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Sor Angela de la Cruz, 24-7.º G. Teléf. 279 40 36. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28020 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América. 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-030-8.
ISBN de la obra: 84-7688-018-9.

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-36080-1986.
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Sor Angela de la Cruz, 24-7.° G. Teléf. 279 40 36. 28020 Madrid

Octubre, 1986.
P.V.P. Canarias: 365,-

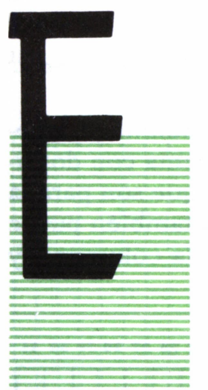
INDICE

1	Conceptos preliminares	5
2	Letras, números y otros personajes	15
3	Los primeros movimientos	29
4	Controlando el movimiento	43
5	Animación de personajes	51
6	Jugando a despistar	59
7	Efectos especiales	69
8	Movimiento en alta resolución	81
	Apéndice	89

Los programas que aparecen en este libro funcionan en los ordenadores:

IBM–PC, XT, AT y compatibles.
AMSTRAD–464, 664, 6128, 1512.
SINCLAIR–SPECTRUM 48 K, 128 K, PLUS, PLUS 2.
MSX–Todos los modelos.
COMMODORE–CBM 64 y CBM 128.

CONCEPTOS PRELIMINARES 1



DESCRIPCION DE LA PANTALLA

EL mundo que nos rodea está en continuo movimiento, y muchas veces se nos ha pasado por la mente la idea de trasladar ese mundo de movimiento y de color a la pantalla de nuestro ordenador. ¿Quién no ha pensado alguna vez en diseñar sus propias aventuras? Esto no es tan complicado como puede parecer a primera vista. Sólo necesitamos tener claros unos cuantos conceptos. En primer lugar, es fundamental conocer el campo en el que se va a producir el movimiento, es decir, la pantalla del ordenador. Por tanto, en este capítulo vamos a adentrarnos un poco en el mundo de la pantalla. En primer lugar, hay que decir que la mayoría de los ordenadores disponen de dos tipos de pantalla: la pantalla de texto o de baja resolución y la pantalla gráfica o de alta resolución. El presente libro se va a ocupar principalmente de los movimientos sobre la pantalla de baja resolución, ya que son los más interesantes de cara al diseño de videojuegos, por tanto, en este capítulo vamos a centrarnos en la descripción de la pantalla de texto.

Comencemos por decir que la pantalla de baja resolución está constituida por una retícula que la divide en «cuadritos», llamados posiciones, ordenados en filas y columnas. Por supuesto, esta retícula es imperceptible por parte del usuario. Las filas y las columnas están numeradas para poder distinguirlas unas de otras, de modo que la pantalla puede asimilarse a un sistema de ejes cartesianos, aunque el origen de coordenadas está en el ángulo superior izquierdo. Por otra parte, la numeración de filas y columnas comienza en unos ordenadores por el cero y en otros por el uno. En resumen, una posición cualquiera de la pantalla vendrá determinada por dos coordenadas: el número de fila y el número de columna.

Sin embargo, el número de filas y de columnas varía de un ordenador a otro; por tanto, vamos a estudiar cada una de las pantallas de las que disponen los ordenadores utilizados en este libro.

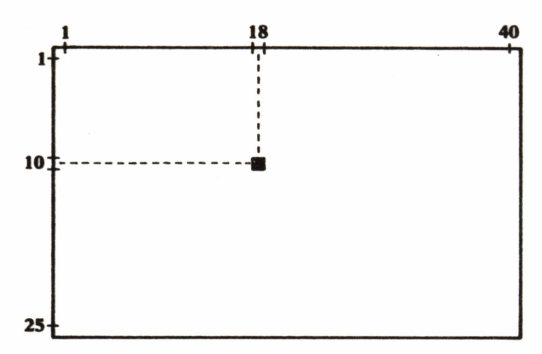


Fig. 1.1. Una posición de la pantalla queda determinada por la fila y la columna.

El Spectrum dispone de una pantalla de 22 filas y 32 columnas, numeradas del 0 al 21 y del 0 al 31 respectivamente. Además cuenta con dos filas más, aunque no podemos utilizarlas para imprimir, ya que son para la edición de líneas de programa. Por otra parte, la pantalla también puede dividirse en dos zonas verticales, la primera comienza en la columna 0 y la segunda en la 16.

Fig. 1.2. Pantalla de texto del Spectrum.

El Amstrad cuenta con tres pantallas de texto, cada una de las cuales se identifica con un número (0, 1 ó 2). Las tres tienen el mismo número de filas, 25, pero se diferencian en el número de columnas. La pantalla 0 cuenta con 20 columnas, la pantalla 1, la más usual, dispone de 40 columnas y la pantalla 2 alcanza las 80 columnas. El Amstrad comienza la numeración de filas y columnas por el 1 en los tres casos. Además las pantallas también pueden dividirse en zonas verticales de 13 columnas como mínimo cada una, lo que significa que la pantalla 0 sólo puede tener una zona que abarca toda la pantalla, mientras que la pantalla 1 dispone de 3 zonas de 13, 13 y 14 columnas respectivamente, y la pantalla 2 cuenta con 6 zonas de 13 columnas cada una, excepto la última que abarca 15 columnas.

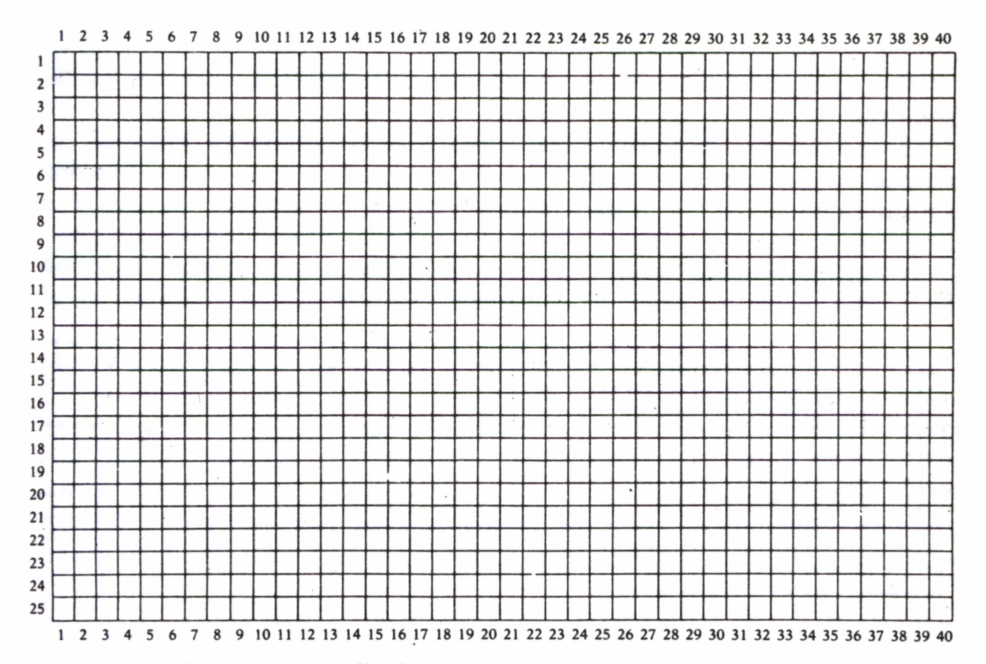


Fig. 1.3. Pantalla de texto de 40 columnas del Amstrad.

Para pasar de una pantalla a otra el Amstrad utiliza la sentencia:

MODE N

donde *N* es un número, 0, 1 ó 2, que indica la pantalla que deseamos. MODE puede utilizarse como comando directo o como instrucción en una línea de programa.

```
10 REM ****************************
20 REM * LAS PANTALLAS DEL AMSTRAD *
30 REM ****************************
40 CLS
50 FOR N=0 TO 2
60 MODE N
70 PRINT "AMSTRAD"
80 PRINT:PRINT
90 PRINT "MODOS DE PANTALLA"
100 PRINT:PRINT
110 PRINT "ESTE ES EL MODO ";N
120 FOR I=1 TO 5000:NEXT
130 NEXT
```

Para MSX e IBM, ver apéndice B.

El programa 1.1 muestra las diferencias de impresión entre una pantalla y otra.

El Commodore dispone de una pantalla de texto con 25 filas y 40 columnas, al igual que la pantalla 1 del Amstrad, sólo que en este ordenador la numeración es de 0 a 24 para las filas y de 0 a 39 para las columnas. Por otra parte, cuenta con cuatro zonas verticales de 10 columnas cada una.

Por último, el IBM tiene dos tipos de pantalla de texto. Ambas disponen de 25 filas, pero una tiene 40 columnas y la otra 80. La numeración, al

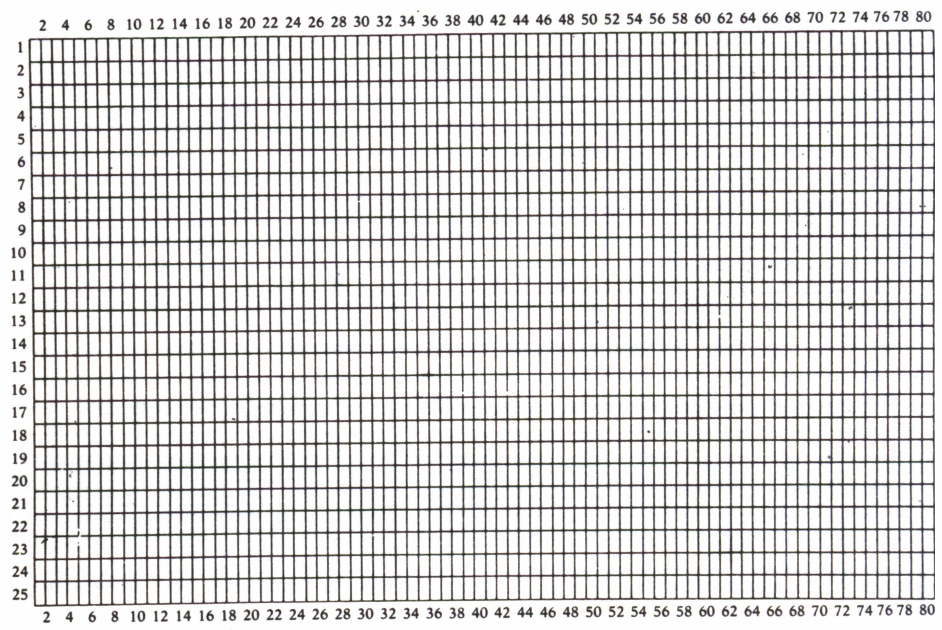


Fig. 1.4. Pantalla de texto de 80 columnas del IBM.

igual que en el Amstrad, comienza por 1, en ambos casos. En cuanto a las zonas verticales el IBM divide sus pantallas en zonas de 14 columnas como mínimo, lo que significa que la pantalla de 40 columnas cuenta con dos zonas que comienzan en las columnas 1 y 15 respectivamente, mientras que la pantalla de 80 columnas tiene cinco zonas de 14 columnas cada una, excepto la última que abarca 24 columnas.

Veamos ahora cómo podemos obtener cada una de las dos pantallas del IBM. Cuando encendemos el ordenador aparece la pantalla de texto de 80 columnas. Sin embargo, puede suceder que nos encontremos el ordenador encendido con una pantalla gráfica. Para pasar a la pantalla de texto utilizaremos la sentencia:

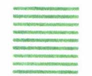
SCREEN 0

Para seleccionar el ancho de pantalla deseado el IBM dispone de la sentencia:

WIDTH A

donde *A* valdrá 40 u 80, según el número de columnas que queramos.

Ya estamos familiarizados con las distintas pantallas de texto, sólo resta puntualizar que a lo largo del libro utilizaremos principalmente la pantalla de 40 columnas en el caso de ordenadores con distintos tipos de pantallas. Sin embargo, podemos utilizar los mismos programas en otros anchos de pantalla sin más que introducir unos sencillos cambios, únicamente en los parámetros que se refieran a números de columna.



SITUACION EN LA PANTALLA

Vamos a comenzar por hacer un breve repaso de la función de los separadores: el punto y coma (;) y la coma (,). Si queremos imprimir en pantalla una serie de datos, ya sean números, cadenas o variables, podemos incluir todos estos datos en una misma sentencia PRINT, separándolos con puntos y comas (;). Este separador hace que los datos se impriman en pantalla uno inmediatamente a continuación del anterior, en la misma línea. Si sustituimos los puntos y comas por comas (,) la impresión se realiza también en la misma línea, pero esta vez cada dato aparece en una zona de la

pantalla (explicadas anteriormente), saltando a la línea siguiente si hubiera más datos que zonas.

```
 10 REM ***********************
 20 REM * COMA Y PUNTO Y COMA *
 30 REM *      SPECTRUM       *
 40 REM ***********************
 50 FOR I=1 TO 160
 60 PRINT "+";"-";
 70 NEXT I
 80 PRINT : PRINT
 90 FOR I=1 TO 10
100 PRINT "-","+",
110 NEXT I
```

Vale para todos los ordenadores.

El programa 1.2 muestra la diferencia entre la utilización del punto y coma, en el primer bucle **FOR-NEXT**, y la coma, en el segundo bucle **FOR-NEXT**. El resultado de la ejecución queda representado en la figura 1.5

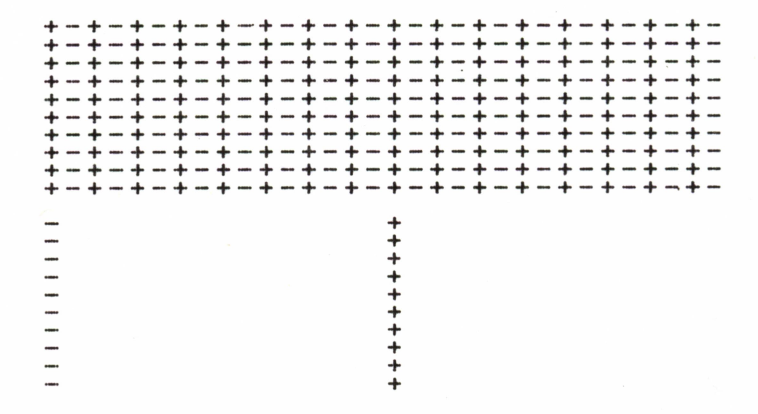


Fig. 1.5. Diferencias de impresión utilizando punto y coma o coma.

Con lo visto hasta ahora, sacamos en conclusión que, de momento, la primera impresión en pantalla se realiza en la primera fila y en la primera columna (suponemos que la pantalla está «limpia». Ahora vamos a ver el modo de imprimir a partir de la columna que nosotros queramos. Para

ello, los ordenadores referidos disponen de la función TAB, que trabaja asociada al PRINT de la siguiente forma:

PRINT	TAB(X);	Número Cadena Variable Expresión	n veces

donde el parámetro *X* representa el número de columna a partir del cual deseamos realizar la impresión. El exponente *n* indica que se puede utilizar varias veces la función TAB dentro de un mismo PRINT, separando unas de otras mediante punto y coma.

```
10 REM **********************
20 REM *    FUNCION TAB     *
30 REM *        IBM         *
40 REM **********************
50 SCREEN 0:WIDTH 40
60 KEY OFF:CLS
70 FOR I=1 TO 22
80 COLOR 14,1,9
90 PRINT TAB(16);"########"
100 NEXT
```

También sirve para MSX e IBM.

El programa 1.3 es un ejemplo del funcionamiento de TAB. El resultado en pantalla queda reflejado en la figura 1.6. Como podemos observar, mediante esta función podemos centrar una figura en la pantalla.

Sin embargo, con la función TAB tenemos que empezar a imprimir siempre en la primera línea. Resulta mucho más interesante poder elegir la fila y la columna en las que queremos realizar la impresión. Esto es posible mediante las funciones AT, para el Spectrum, y LOCATE, en los demás ordenadores.

La función AT, al igual que TAB, se utiliza dentro de un PRINT y tiene el siguiente formato:

PRINT	AT Y,X;	Número Cadena Variable Expresión	n veces

```
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
########
```

Fig. 1.6. Impresión en pantalla utilizando la función TAB.

donde *Y* representa la fila y *X* la columna que indican la posición de la pantalla donde se va a realizar la impresión. Dentro de un PRINT podemos incluir varios AT separándolos con punto y coma.

```
10 REM **************
20 REM * FUNCION AT *
30 REM *  SPECTRUM  *
40 REM **************
50 FOR I=0 TO 21
60 PRINT AT I,I+5;"\";AT I,26-I;"/"
70 NEXT I
```

El programa 1.4 dibuja un aspa en el centro de la pantalla del Spectrum mediante la función AT. El resultado está representado en la figura 1.7.

La función LOCATE tiene el mismo objetivo que AT, sólo que constituye una instrucción independiente del PRINT, aunque normalmente se sitúa delante de él.

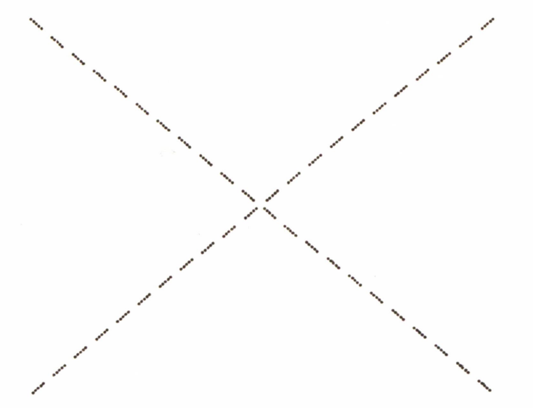


Fig. 1.7. Aspa obtenida utilizando la función AT.

El formato más común es el siguiente:

LOCATE X,Y: PRINT	Número Cadena Variable Expresión

donde *X* es el número de columna e *Y* es el número de fila que determinan la posición de la pantalla en la que se va a imprimir. El IBM tiene la particularidad de que el orden de los parámetros es inverso, es decir, la forma es: LOCATE Y,X

```
10 REM *****************
20 REM * FUNCION LOCATE *
30 REM *   COMMODORE    *
40 REM *****************
50 PRINT CHR$(147)
60 FOR I=2 TO 22 STEP 4
70 LOCATE I+8,I:PRINT "*"
80 NEXT I
```

Ver apéndice B.

El programa 1.5 es un ejemplo de las posibilidades de LOCATE. El resultado en pantalla es el representado en la figura 1.8.

Conviene advertir que el Commodore solo dispone de la función LO-

```
        *

           *

              *

                 *

                    *

                       *
  OK
```

Fig. 1.8. Pantalla obtenida mediante la función LOCATE.

CATE en la versión 4.5 del BASIC, que es una extensión del BASIC que se encuentra en los antiguos Commodore.

Estas son las funciones fundamentales que necesitamos conocer para poder empezar a realizar programas llenos de movimiento, por tanto, sigamos adelante.

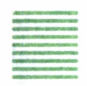
LETRAS, NUMEROS Y OTROS PERSONAJES 2

INTRODUCCION





ANTES de introducirnos en el mundo de la simulación de movimientos con ordenador vamos a ver qué podemos mover por la pantalla. En principio podemos producir el movimiento de todo aquello que podamos imprimir en pantalla, es decir, desde una simple letra, como la A, hasta cualquier personaje, animal o marciano que se nos ocurra, como por ejemplo, un caballo. Evidentemente este último caso es un poco más complejo que el primero, sobre todo si tenemos en cuenta que todos los ordenadores tienen una tecla con la letra A, por lo que resulta muy fácil imprimirla, mientras que sería bastante raro encontrar un ordenador que tuviera un caballo en una tecla. Sin embargo, todo es posible, pero vamos a ir por partes.



EL JUEGO DE CARACTERES Y EL CODIGO ASCII

Todos los ordenadores disponen de un conjunto de caracteres ya definidos en origen, y que podemos imprimirlos en pantalla con solo apretar la tecla correspondiente o, a lo sumo, dos teclas simultáneamente. Este conjunto de caracteres está constituido por letras (mayúsculas y minúsculas), números y signos especiales como los que encontramos en cualquier máquina de escribir (punto, asterisco, paréntesis, signo igual, etc.). Además en el juego de caracteres de cualquier ordenador podemos encontrar también un conjunto de caracteres gráficos (cuadrados, rectángulos, triángulos, rombos, etc.), que será más o menos variado dependiendo del fabricante.

El número máximo de caracteres distintos que puede tener un ordenador es de 256. Cada uno de estos caracteres predefinidos se almacena

en la memoria del ordenador en un byte o celda de memoria. Cada byte está constituido por ocho bits y en cada bit sólo puede almacenarse un 0 o un 1.

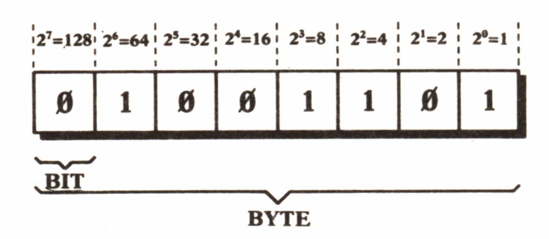


Fig. 2.1. Esquema de una celda de memoria o byte con un número binario almacenado.

Entonces, ¿cómo es posible almacenar un carácter cualquiera, como, por ejemplo, una letra, en un byte? La respuesta es sencilla; cada carácter predefinido tiene un número asociado que lo representa; por tanto, si puede haber 256 caracteres distintos, dicho número estará comprendido entre 0 y 255. Esta relación entre los caracteres predefinidos y sus códigos numéricos asociados constituye el denominado código ASCII. Al final del libro se incluye un apéndice con los códigos ASCII correspondientes a cada uno de los ordenadores que se estudian en el presente libro. Sin embargo, aunque ya sabemos que un carácter cualquiera se almacena en memoria representado por su código ASCII correspondiente, no parece que un número entre 0 y 255 sea lo mismo que la secuencia de 0 y 1 que se almacena realmente en un byte, pero en realidad sí es lo mismo.

Lo que sucede es que la secuencia de 0 y 1 que se almacena en un byte representa un número en notación binaria. Para saber el número decimal representado en binario no tenemos más que sumar los números correspondientes a cada bit en el que haya almacenado un 1. Así, si queremos averiguar el número decimal almacenado en el byte de la figura 2.1, la transformación será la siguiente:

$$\underset{\text{BINARIO}}{\underline{01001101}} = 2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = \underset{\text{DECIMAL}}{\underline{77}}$$

Obtenemos el número 77 que es el código ASCII correspondiente a la letra M mayúscula. Ahora bien, existen dos funciones, inversas entre sí, que nos permiten conocer el código ASCII asociado a un carácter, y a la

inversa, el carácter correspondiente a un código determinado. Estas funciones son ASC (o CODE) y CHR$.

La función ASC (CODE en el Spectrum) tiene el formato siguiente:

```
ASC ("carácter")
```

y se encarga de transformar el *carácter* que aparece como argumento, en el número del código ASCII que le corresponde. Así, por ejemplo, si tecleamos:

```
PRINT ASC ("A")
```

en pantalla aparecerá el número 65.

La función CHR$, más útil para el tema de este libro, tiene el siguiente formato:

```
CHR$ (N)
```

donde *N* es un número entre 0 y 255. La función CHR$ transforma dicho número en el carácter correspondiente, según la equivalencia del código ASCII.

Si, por ejemplo, tecleamos el comando:

```
PRINT CHR$(42)
```

obtendremos en pantalla un asterisco.

La función CHR$ nos va a resultar muy útil, sobre todo para imprimir caracteres gráficos que, en muchas ocasiones, no vienen representados en el teclado.

```
10 REM ****************
20 REM * CODIGO ASCII *
30 REM ****************
```

```
40 CLS
50 FOR I=32 TO 255
60 PRINT I,CHR$(I)
70 NEXT I
```

Para Commodore, ver apéndice C.

El programa 2.1 permite la visualización en pantalla del juego de caracteres definidos en el ordenador que estemos utilizando.

Por último, conviene señalar que los caracteres más estándar como son letras, números y signos (de puntuación, aritméticos, etc.), tienen siempre el mismo código ASCII en todos los ordenadores. Estos códigos ya establecidos varían normalmente entre 32 (espacio en blanco) y 126 (signo ~), excepto en el Commodore que empieza a incluir caracteres gráficos a partir del código 96.

Los primeros códigos suelen estar asociados a teclas de control (ENTER, flechas de movimiento de cursor, CTRL, etc.) o caracteres especiales. Los códigos finales normalmente se asocian a caracteres gráficos y especiales, que pueden variar bastante de un ordenador a otro. La única excepción la presenta el Spectrum que asigna sus últimos códigos ASCII a las palabras reservadas del BASIC, aunque éstas no sean realmente caracteres.



DEFINIENDO NUEVOS CARACTERES

Ya dijimos al principio del capítulo que podíamos simular el movimiento de cualquier personaje que se nos ocurriera, pero ¿qué sucede cuando el personaje en cuestión no está definido en el juego de caracteres del ordenador? En principio, no podríamos imprimirlo en la pantalla y por consiguiente no podría desplazarse por ella. Sin embargo, muchos ordenadores nos permiten la posibilidad de definir nuestros propios caracteres. Para ello utilizaremos normalmente códigos asociados a otros caracteres (generalmente gráficos) que no vayamos a utilizar y que podemos redefinir, eliminando el antiguo carácter e introduciendo en su lugar el carácter que nosotros deseemos.

Esto es muy interesante, así que vamos a ver cómo se define un carácter. Cada posición de la pantalla de baja resolución (o pantalla de texto) está constituida por una malla de 8 x 8 puntos, lo que significa que cuando imprimimos un carácter cualquiera sobre una posición de la pantalla, lo que estamos haciendo realmente es representar el dibujo de dicho carácter sobre la mencionada malla de 8 x 8. Cada uno de los caracteres definidos están representados en una malla de 8 × 8, de modo que los puntos que constituyen el dibujo del carácter serán puntos encendidos de la ma-

lla y serán los que se impriman en pantalla, mientras que el resto de los puntos permanecen apagados y se imprimen en pantalla con el mismo color del fondo (o papel).

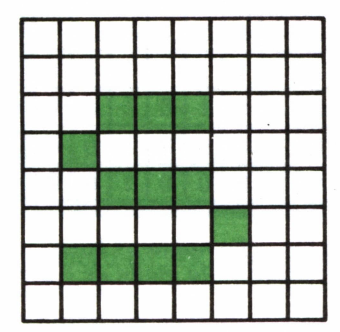


Fig. 2.2. Representación de la s minúscula en la malla de 8 por 8 puntos.

La figura 2.2 muestra la representación de la letras minúsculas en la malla de 8 por 8. Los puntos en negro son losencendidos, mientras que el resto están apagados. Pues bien, del mismo modo nosotros podemos representar en la malla de 8 por 8 cualquier figura que deseemos, convirtiéndola así en un carácter.

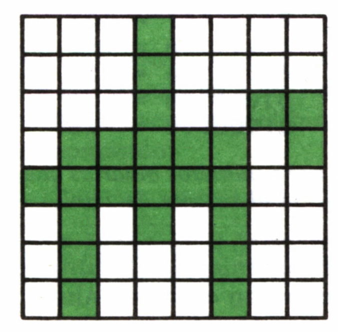


Fig. 2.3. Representación de caballo con jinete en la malla de 8 por 8 puntos.

En la figura 2.3 hemos diseñado un nuevo carácter que representa un caballo con jinete. Veamos ahora cómo le podemos introducir al ordenador nuestro nuevo carácter. La malla de 8 por 8 está representada en la memoria del ordenador por un conjunto de 8 bytes. Recordemos que cada byte está constituido por 8 bits por tanto nuestra malla de 8 por 8 puntos está representada por un conjunto de 64 bits. Para indicar los puntos encendidos que constituyen un carácter, en cada bit correspondiente habrá almacenado un 1, mientras que los puntos apagados están representados por 0.

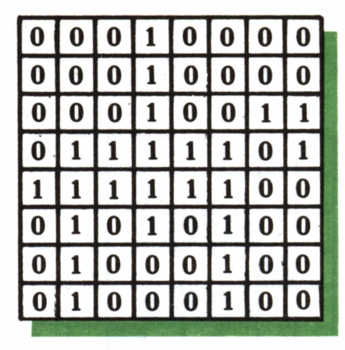


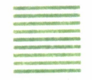
Fig. 2.4. Representación de los ocho números binarios necesarios para definir un caballo con jinete en la malla de 8 por 8 puntos.

La figura 2.4 muestra cómo quedaría nuestro caballo con jinete representado con ceros y unos. En realidad está constituido por un conjunto de 8 bytes en cada uno de los cuales hay almacenado un número binario, que podríamos transformar a decimal mediante el método que ya conocemos o consultando la tabla de conversión que se incluye al final del libro.

```
00010000 =  16
00010000 =  16
00010011 =  19
01111101 = 125
11111100 = 252
01010100 =  84
01000100 =  68
01000100 =  68
```

Fig. 2.5. Conversión de los ocho números binarios que definen el caballo con jinete al sistema decimal.

En la figura 2.5 hemos obtenido los 8 números decimales que constituyen el carácter del caballo con el jinete. Ahora sólo tenemos que introducirlos en la memoria del ordenador en el lugar ocupado por otro carácter que no vayamos a utilizar. El modo de introducir estos números en la memoria varía mucho de un ordenador a otro por tanto vamos a estudiarlo para cada ordenador separadamente.



DEFINICION DE CARACTERES EN EL SPECTRUM

Sabemos que cada byte de la memoria de un ordenador está representado por un número que es su dirección de memoria por tanto para introducir los ocho números que definen el nuevo carácter en los ocho bytes

correspondientes tenemos que dirigirnos a ocho direcciones de memoria consecutivas. En el Spectrum podemos definir 21 caracteres nuevos, asignando cada uno de ellos a una tecla comprendida entre la A y la U. Para averiguar las direcciones de memoria asignadas a cada tecla para la definición de caracteres, el Spectrum cuenta con la función USR. Si tecleamos el comando:

```
PRINT USR "A"
```

en pantalla aparecerá el número 64349, que es precisamente la dirección de memoria correspondiente al primer byte en el que pueden definirse nuevos caracteres. Si añadimos las siete direcciones de memoria siguientes tendremos el conjunto de ocho bytes necesario para introducir un nuevo carácter en la tecla de la letra A. Para introducir un número (entre 0 y 255) en una dirección de memoria utilizaremos la sentencia POKE, que tiene el siguiente formato:

```
POKE DM, N
```

donde *DM* es una dirección de memoria y *N* es un número entre 0 y 255. Si el número que vamos a introducir está en notación binaria, tenemos que indicarlo con la función BIN seguida del binario correspondiente. Si tecleamos, por ejemplo, el comando:

```
POKE 28145, BIN 01001010
```

habremos introducido en el byte de dirección de memoria 28145 el binario 01001010, que en sistema decimal representa el número 74.

Por otra parte, la instrucción PEEK se encarga de buscar lo que hay almacenado en una dirección de memoria determinada; por tanto, si queremos comprobar la operación anterior no tenemos más que teclear el siguiente comando:

```
PRINT PEEK (28145)
```

y en pantalla aparecerá efectivamente el 74. Si lo que queremos es ver el carácter correspondiente al número almacenado en esa posición de memoria, teclearemos el comando:

```
PRINT CHR$(PEEK(28145))
```

y obtendremos en pantalla una J mayúscula.

Bueno, pues ya conocemos todas las sentencias y funciones necesarias para definir un nuevo carácter; por tanto, podemos ver que el programa 2.2 se encarga de definir nuestro caballo con jinete de la figura 2.3 en la tecla A del Spectrum.

```
 10 REM **********************
 20 REM * CABALLO CON JINETE *
 30 REM *      SPECTRUM      *
 40 REM **********************
 50 POKE USR "A"+0,BIN 00010000
 60 POKE USR "A"+1,BIN 00010000
 70 POKE USR "A"+2,BIN 00010011
 80 POKE USR "A"+3,BIN 01111101
 90 POKE USR "A"+4,BIN 11111100
100 POKE USR "A"+5,BIN 01010100
110 POKE USR "A"+6,BIN 01000100
120 POKE USR "A"+7,BIN 01000100
```

Si queremos introducir los números en notación decimal en lugar de en binario, no tenemos más que suprimir la función BIN y sustituir cada número binario por su decimal correspondiente (calculados en la figura 2.5).

Para poder visualizar por fin nuestro caballito en pantalla, primero tenemos que ejecutar el programa. A continuación tenemos que pasar el teclado a modo gráfico (pulsando simultáneamente CAPS SHIFT y GRAPHICS). Ahora cada vez que pulsemos la tecla de la A (en modo gráfico) aparecerá en pantalla nuestro nuevo carácter tal y como aparece en la figura 2.6.



Fig. 2.6. Este caballito queda definido en la tecla A del Spectrum (en modo gráfico), después de ejecutar el programa 2.2.



DEFINICION DE CARACTERES EN EL AMSTRAD

Para definir nuestros propios caracteres en el Amstrad no son necesarias ocho líneas de programa como en el Spectrum, ya que contamos con la sentencia SYMBOL, que tiene el siguiente formato:

SYMBOL C (lista de ocho números)

donde *C* es el número del código ASCII asociado al carácter que vamos a redefinir y la *lista de ocho números* está constituida por los ocho números, en notación decimal o hexadecimal, que representan los ocho binarios necesarios para definir el carácter deseado.

Por tanto, para definir el caballo con jinete de la figura 2.3 no tenemos más que teclear el programa 2.3.

```
10 REM ***********************
20 REM * CABALLO CON JINETE *
30 REM *        AMSTRAD       *
40 REM ***********************
50 MODE 1:CLS
60 SYMBOL 240,16,16,19,125,252,84,68,68
70 LOCATE 20,13:PRINT CHR$(240)
```

Para MSX, ver apéndice B.

Los ocho números decimales los obtuvimos en la figura 2.5. La línea 70 del programa se encarga de imprimir nuestro caballito en el centro de la pantalla.

La instrucción SYMBOL nos permite definir nuevos caracteres a partir del código 240, por tanto podemos redefinir 16 nuevos caracteres. Sin embargo, tenemos la posibilidad de definir un nuevo carácter en cualquier otro código gracias a la sentencia SYMBOL AFTER, que tiene el siguiente formato:

SYMBOL AFTER C

donde *C* es un número que indica el nuevo código a partir del cual podemos redefinir caracteres.

```
10 REM ***************
20 REM * HOMBRECILLO *
30 REM *   AMSTRAD   *
40 REM ***************
50 MODE 1:CLS
60 SYMBOL AFTER 90
70 SYMBOL 106,24,24,126,153,189,36,36,102
80 FOR I=1 TO 39 STEP 2
90 LOCATE I,13:PRINT CHR$(106)
100 NEXT
```

El programa 2.4 posibilita la redefinición de caracteres a partir del código 90 (correspondiente a la Z mayúscula). En la línea 70 definimos un hombrecito en el carácter de código 106 (j minúscula). El bucle FOR-NEXT de las líneas 80 a 100 permite imprimir 20 hombrecitos en la fila central de la pantalla, tal y como muestra la figura 2.7.

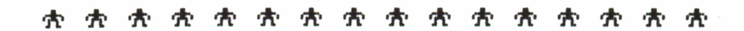


Fig. 4.7. Hombrecito definido con el código ASCII 106 (j mayúscula) en el AMSTRAD.

Sin embargo, de este modo perdemos la posibilidad de imprimir la j minúscula en pantalla, ya que al redefinir un nuevo carácter en el Amstrad perdemos el caracter que tuviera asignado ese código anteriormente.



DEFINICION DE SPRITES EN COMMODORE

La definición de nuevos caracteres en Commodore resulta bastante compleja, ya que no dispone de instrucciones **BASIC** para resolver el problema. Sin embargo, el Commodore admite la generación de los llamados «Sprites», que permiten el diseño de figuras sobre una malla de 24 por 21 puntos, lo que nos da, dividiendo por ocho bits, un total de 63 bytes, en cada uno de los cuales hay que almacenar un número binario.

La figura 2.8 muestra un ejemplo de diseño de un sprite con la figura de un invasor galáctico. Cada fila de la malla tiene tres bytes (24 bits), por tanto, cada punto negro hay que sustituirlo por 1, mientras que los puntos en blanco se sustituyen por 0, tal y como muestra la figura 2.9.

Por último, tenemos que transformar cada número binario a notación decimal, para poder introducir cada uno de los valores obtenidos en la par-

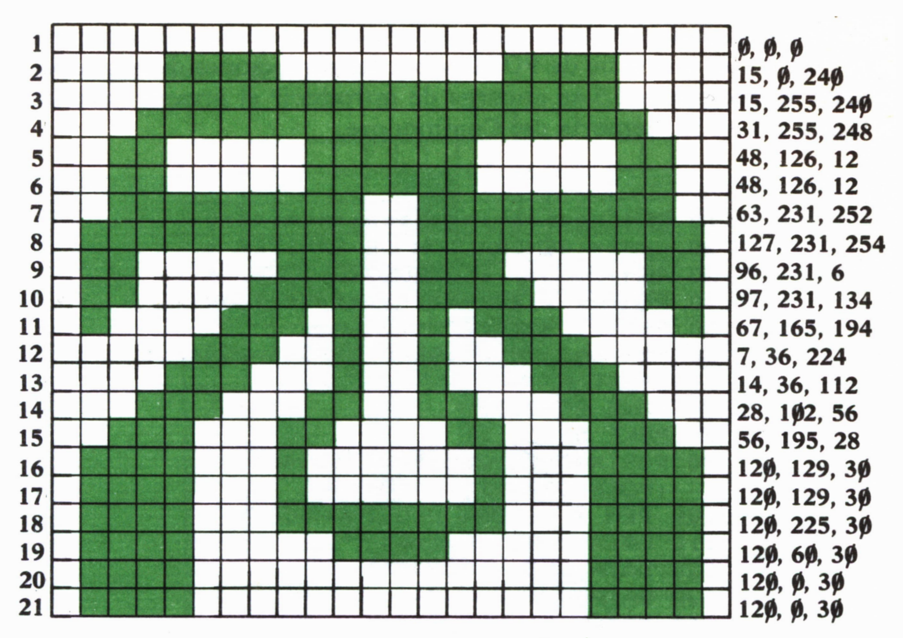


Fig. 2.8. Definición de un invasor galáctico en un Sprite del Commodore.

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
3	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
4	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
5	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0	0
6	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0	0
7	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0
8	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0
9	0	1	1	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	0
10	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0	0	1	1	0
11	0	1	0	0	0	0	1	1	1	0	1	0	0	1	0	1	1	1	0	0	0	0	1	0
12	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0
13	0	0	0	0	1	1	1	0	0	0	1	0	0	1	0	0	0	1	1	1	0	0	0	0
14	0	0	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	0	0
15	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	1	1	0	0
16	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0
17	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0
18	0	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	0
19	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0
20	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
21	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0

Fig. 2.9. Representación del invasor galáctico en código de ceros y unos.

te de la memoria reservada para la generación de sprites. Esta generación se resuelve con una serie de **POKE** sobre las direcciones de memoria adecuadas. El programa 2.5 se encarga de introducir el invasor galáctico de la figura 2.8 en la memoria del Commodore.

```
10 REM *************************
20 REM * GENERACION DE SPRITES *
30 REM *       COMMODORE       *
40 REM *************************
50 PRINT CHR$(147)
60 FOR S=0 TO 62:READ D:POKE 832+S,D
   :NEXT S
70 POKE 2040,13
80 POKE 53269!,1
90 FOR X=0 TO 255:POKE 53248!,X
   :POKE 53249!,100:NEXT
100 REM * FIGURA DEL INVASOR *
110 DATA   0,  0,  0
120 DATA  15,  0,240
130 DATA  15,255,240
140 DATA  31,255,240
150 DATA  48,126, 12
160 DATA  48,126, 12
170 DATA  63,231,252
180 DATA 127,231,254
190 DATA  96,231,  6
200 DATA  97,231,134
210 DATA  67,165,194
220 DATA   7, 36,224
230 DATA  14, 36,112
240 DATA  28,102, 56
250 DATA  56,175, 28
260 DATA 120,129, 30
270 DATA 120,129, 30
280 DATA 120,255, 30
290 DATA 120, 60, 30
300 DATA 120,  0, 30
310 DATA 120,  0, 30
```

Los números decimales almacenados en las líneas **DATA** son los que constituyen los 63 bytes necesarios para la generación del Sprite.

Vista ya la definición de caracteres en tres ordenadores distintos, podríamos, evidentemente, construir figuras más grandes uniendo distintos caracteres nuevos, como se muestra en la figura 2.10.

Finalmente, conviene decir que aunque la definición de nuevos caracteres resulta interesante, no es imprescindible, ya que la mayoría de las fi-

guras que inventemos podemos construirlas combinando algunos de los caracteres gráficos ya definidos.

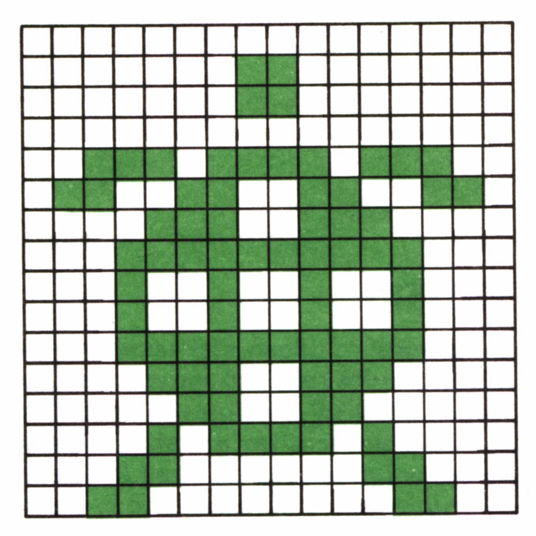


Fig. 2.10. Diseño de una tortuga utilizando varios caracteres definidos por el usuario.

LOS PRIMEROS MOVIMIENTOS 3

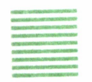


INTRODUCCION



PARA empezar este capítulo vamos a tratar, en primer lugar, de hacernos una idea sobre cómo se produce el efecto de movimiento en la pantalla del ordenador. Sobre la pantalla podemos ver moverse desde una pequeña y simple pelota hasta un terrible monstruo que persigue al personaje protagonista de una aventura. Todas estas figuras, aunque dispares, se basan en la misma idea para moverse. Dicha idea se fundamenta en un principio similar a los dibujos animados: se dibuja el personaje en una posición determinada y en el fotograma siguiente se le dibuja en una posición distinta, aunque próxima, de modo que, repitiendo la operación para miles de fotogramas, se consigue que el personaje se mueva de un lado a otro al proyectar la película. Pues bien, si trasladamos esta idea a ordenador tendremos que imprimir la figura que va a moverse en una posición determinada de la pantalla, y seguidamente, borrarla de esa posición para imprimirla en una posición próxima, de modo que el efecto que se produce es que la figura se ha trasladado de una posición a otra. Por otra parte, la figura que vamos a mover puede estar compuesta por un solo carácter predefinido, como por ejemplo una pelota que puede representarse con la letra o, pero también puede ser una figura compleja compuesta por varios caracteres, predefinidos o definidos por nosotros mismos, en cuyo caso tendremos que tener cuidado de imprimir cada uno de ellos en la posición correcta para obtener la figura deseada.

Bueno pues, vistas las primeras ideas generales, podemos empezar ya con los primeros movimientos.



MOVIMIENTO DE UN CARACTER SOBRE UNA RECTA

Vamos a empezar por estudiar cómo podríamos conseguir el efecto de movimiento de un carácter cualquiera, por ejemplo un asterisco, avanzan-

do a lo largo de una recta horizontal. En este tipo de movimiento el carácter se imprime en una posición inicial cualquiera determinada por su fila y su columna. A continuación sólo hay que borrarlo e imprimirlo seguidamente en la misma fila, pero en la columna siguiente; por tanto, la fila permanecerá siempre constante, mientras que la columna irá aumentando de uno en uno.

```
 10 REM ************************
 20 REM *MOVIMIENTO HORIZONTAL*
 30 REM * IZQUIERDA - DERECHA *
 40 REM *       SPECTRUM       *
 50 REM ************************
 60 LET X=11: LET Y=0
 70 CLS
 80 PRINT AT X,Y;"*"
 90 LET Y=Y+1
100 IF Y>31 THEN  LET Y=0
110 PAUSE 5
120 GO TO 70
```

El programa 3.1 produce el movimiento de un asterisco de izquierda a derecha por el centro de la pantalla. Pero, evidentemente, la pantalla tiene unos límites; por tanto, ¿qué sucede cuando el asterisco llega al extremo izquierdo? Pues que desaparece y vuelve a aparecer por la parte derecha para seguir avanzando. Parece como si hubiera dado la vuelta por detrás del televisor para volver a aparecer en la pantalla. Este efecto se consigue gracias a la condición de la línea 100 que controla cuándo el asterisco está en la última columna de la pantalla (columna 31 en el Spectrum). Cuando se verifica esta situación el asterisco desaparece de la posición en la que estaba para imprimirse otra vez en la columna inicial (columna 0 en el Spectrum). Si no pusiéramos la condición de la línea 100, cuando llegara el asterisco al final de la pantalla nos daría un mensaje de error, ya que el ordenador intentaría imprimir en una columna que no existe.

```
10 REM ****************************
20 REM *   MOVIMIENTO HORIZONTAL   *
30 REM *  DE IZQUIERDA A DERECHA   *
40 REM *          AMSTRAD          *
50 REM ****************************
60 MODE 1
```

```
70 X=13:Y=1
80 CLS
90 LOCATE Y,X:PRINT "*"
100 Y=Y+1
110 IF Y>40 THEN Y=1
120 FOR I=1 TO 30:NEXT
130 GOTO 80
```

Para MSX, ver apéndice B.

El programa 3.2 tiene el mismo objetivo que el 3.1, sólo que incluye las variaciones para el Amstrad. También sería válido en el IBM y en el Commodore introduciendo unos pequeños matices. En el IBM sólo habría que cambiar el **LOCATE Y,X** de la línea 90 por **LOCATE X,Y**, es decir, invertir el orden de los parámetros (fila, columna, en lugar de columna, fila); también hay que sustituir la línea: 60 **MODE** 1 por la línea: 60 **SCREEN** 0 : **WIDTH** 40, para conseguir el ancho de pantalla de 40 columnas. Por otra parte, en el Commodore hay que sustituir la línea: 80 **CLS** por la línea: 80 **PRINT CHR$**(147), para borrar la pantalla; también tenemos que suprimir la línea 60, ya que el Commodore tiene una única pantalla de 40 columnas, aunque, eso sí, tendremos que variar las instrucciones donde entran en juego los límites de la pantalla, ya que las columnas están numeradas de 0 a 39 en lugar de 1 a 40. Esto significa que en la línea 70 tendríamos que teclear Y=0 en vez de Y=1 y la condición de la línea 110 quedaría sustituida por: 110 IF Y>39 THEN Y=0.

Por último, la velocidad del movimiento podemos variarla a nuestro gusto, sólo con cambiar el valor final del temporizador (bucle **FOR-NEXT**) de la línea 120. Cuanto más grande sea el valor, más despacio se moverá el asterisco.

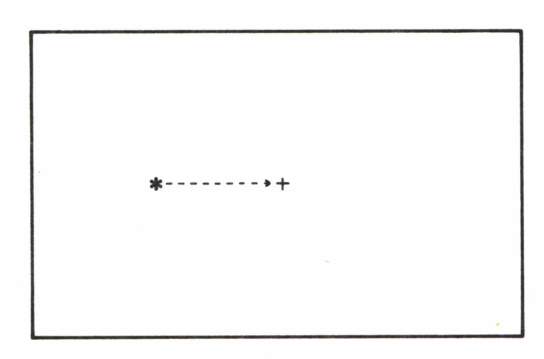


Fig. 3.1. El asterisco se desplaza horizontalmente hacia la derecha.

Bueno, el efecto de movimiento conseguido resulta bastante convincente pero vamos a suponer que el asterisco se desplaza por una calle dibujada en la pantalla. Si ahora utilizamos el mismo esquema de los programas 3.1 ó 3.2, para simular el movimiento resulta que al utilizar la instrucción CLS para borrar el asterisco lo que hacemos es borrar toda la pantalla y por tanto la calle desaparece.

```
 10 REM ************************
 20 REM * MOVIMIENTO POR CALLE*
 30 REM *       SPECTRUM       *
 40 REM ************************
 50 FOR I=0 TO 31
 60 PRINT AT 10,I;CHR$ 129;AT 12,I;CHR$ 132
 70 NEXT I
 80 LET X=11: LET Y=0
 90 PRINT AT X,Y;" *"
100 LET Y=Y+1
110 IF Y>30 THEN  PRINT AT X,Y;" ": LET Y=0
120 PAUSE 5
130 GO TO 90
```

En el programa 3.3 el problema queda solucionado, ya que si lo ejecutamos podremos ver cómo el asterisco avanza por la calle sin que ésta se borre de la pantalla. La solución es sencilla: no es necesario borrar toda la pantalla para hacer desaparecer el asterisco de la posición en la que se encontraba, sino que basta con borrar dicha posición. Esto se consigue imprimiendo encima un espacio en blanco. Este nuevo método para simular movimiento nos va a ser muy útil, ya que si queremos diseñar algún videojuego sencillo éste será el método utilizado, puesto que siempre habrá alguna figura fija en la pantalla que no nos interese borrar.

```
 10 REM ********************************
 20 REM *  MOVIMIENTO POR UNA CALLE    *
 30 REM *   DE IZQUIERDA A DERECHA     *
 40 REM *          COMMODORE           *
 50 REM ********************************
 60 PRINT CHR$(147)
 70 FOR I=0 TO 39
 80 LOCATE I,11:PRINT CHR$(188)
 90 LOCATE I,13:PRINT CHR$(172)
100 NEXT I
110 X=12:Y=0
120 LOCATE Y,X:PRINT " *"
130 Y=Y+1
```

```
140 IF Y>38 THEN LOCATE Y,X:PRINT " ":Y=0
150 FOR I=1 TO 100:NEXT I
160 GOTO 120
```

Ver apéndice C.

El programa 3.4 es análogo al 3.3, pero para el Commodore (pantalla de 40 columnas). En el IBM y el Amstrad tendríamos que introducir unas sencillas variaciones similares a las explicadas para el programa 3.2. En cualquier caso todas estas pequeñas variaciones están referidas en el apéndice A, al final del libro.

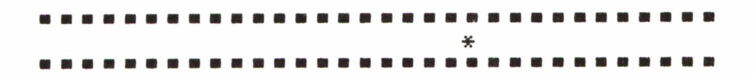


Fig. 3.2. El asterisco parece ahora un insecto paseando por una calle.

Hasta aquí hemos visto el primer movimiento simulado por el ordenador: el desplazamiento horizontal de un asterisco. A continuación vamos a ver el movimiento vertical. Para ello vamos a utilizar de nuevo el método de imprimir un espacio en blanco en la posición de la pantalla que queramos borrar.

```
 10 REM ***********************
 20 REM * MOVIMIENTO VERTICAL *
 30 REM *  DE ARRIBA A ABAJO  *
 40 REM *      SPECTRUM       *
 50 REM ***********************
 60 FOR I=0 TO 21
 70 PRINT AT I,14;CHR$ 138;AT I,16;CHR$ 133
 80 NEXT I
 90 LET X=0: LET Y=15
100 PRINT AT X,Y;" "
110 PRINT AT X+1,Y;"O"
120 LET X=X+1
130 IF X>20 THEN  PRINT AT X,Y;" ": LET X=0
140 PAUSE 5
150 GO TO 100
```

La ejecución del programa 3.5 muestra una pelota (representada por una O mayúscula) cayendo por un tubo. El planteamiento es análogo al del programa 3.3, pero en este caso lo que va aumentando es la fila (línea 120),

en vez de la columna, para que se produzca el movimiento vertical. Naturalmente, en este caso el límite de la pantalla que hay que controlar es el borde inferior, para que el ordenador no trate de imprimir en una línea que no existe (fuera de la pantalla); esto lo conseguimos con la condición de la línea 130.

```
10 REM **************************
20 REM *  MOVIMIENTO VERTICAL   *
30 REM *   DE ARRIBA A ABAJO    *
40 REM *         IBM            *
50 REM **************************
60 SCREEN 0:WIDTH 40
70 KEY OFF:CLS
80 FOR I=1 TO 22
90 LOCATE I,19:PRINT CHR$(221)
100 LOCATE I,21:PRINT CHR$(222)
110 NEXT
120 X=1:Y=20
130 LOCATE X,Y:PRINT " "
140 LOCATE X+1,Y:PRINT "O"
150 X=X+1
160 IF X>21 THEN LOCATE X,Y:PRINT " "
    :X=1
170 FOR I=1 TO 100:NEXT
180 GOTO 130
```

Para el Amstrad cambiar la línea 60 por 'MODE 1' y la 70 por 'CLS'.

El programa 3.6 cumple el mismo objetivo que el 3.5, pero para una pantalla de 25 líneas como la del IBM. En Amstrad y Commodore sería bastante similar.

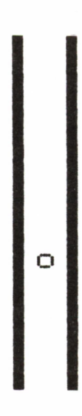


Fig. 3.3. Por el tubo van cayendo sucesivas «pelotitas».

Ya sabemos cómo simular movimientos horizontales y verticales, sólo nos falta controlar el desplazamiento en diagonal. En este caso el programa es muy parecido a los vistos anteriormente, sólo que ahora hay que incrementar simultáneamente filas y columnas, ya que un movimiento en diagonal no es más que la composición de dos desplazamientos: uno vertical (aumentando las filas) y uno horizontal (aumentando las columnas).

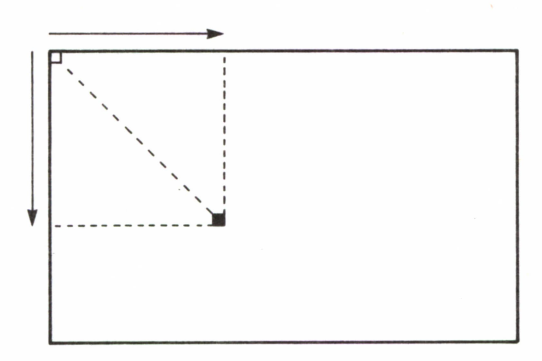


Fig. 3.4. Esquema de movimiento diagonal.

Vamos a suponer un movimiento diagonal hacia abajo y hacia la derecha. Si seguimos la técnica de imprimir un espacio en blanco para borrar la posición anterior, en este caso dicha posición vendrá determinada por una fila menos y una columna menos que la posición en la que se encuentra el carácter que se está deplazando. Esta idea es la que sigue el programa 3.7.

```
 10 REM ************************
 20 REM * MOVIMIENTO DIAGONAL *
 30 REM *       SPECTRUM       *
 40 REM ************************
 50 FOR I=0 TO 20
 60 PRINT AT I,I+1;"\";AT I+1,I;"\"
 70 NEXT I
 80 LET X=0: LET Y=0
 90 PRINT AT X,Y;" "
100 PRINT AT X+1,Y+1;CHR$ 134
110 LET X=X+1: LET Y=Y+1
120 IF X>20 THEN  PRINT AT X,Y;"
    ": LET X=0: LET Y=0
130 PAUSE 5
140 GO TO 90
```

El programa 3.7 produce el movimiento diagonal que queríamos, pero como la pantalla no es cuadrada el carácter que se desplaza llega antes al

margen inferior que al margen derecho por tanto nos basta controlar que las filas no superen el límite de la pantalla (condición de la línea 120). Por lo demás el programa es muy similar a los anteriores.

```
10 REM *************************
20 REM *  MOVIMIENTO DIAGONAL   *
30 REM *        AMSTRAD         *
40 REM *************************
50 MODE 1:CLS
60 FOR I=1 TO 24
70 LOCATE I+1,I:PRINT "\"
80 LOCATE I,I+1:PRINT "\"
90 NEXT
100 X=1:Y=1
110 LOCATE X,Y:PRINT " "
120 LOCATE X+1,Y+1:PRINT CHR$(137)
130 X=X+1:Y=Y+1
140 IF X>24 THEN LOCATE X,Y:PRINT " "
    :X=1:Y=1
150 FOR I=1 TO 100:NEXT
160 GOTO 110
```

Para MSX e IBM, ver apéndice B.

El programa 3.8 es la versión para Amstrad del programa 3.7 por tanto los comentarios hechos para el anterior son válidos para este programa.

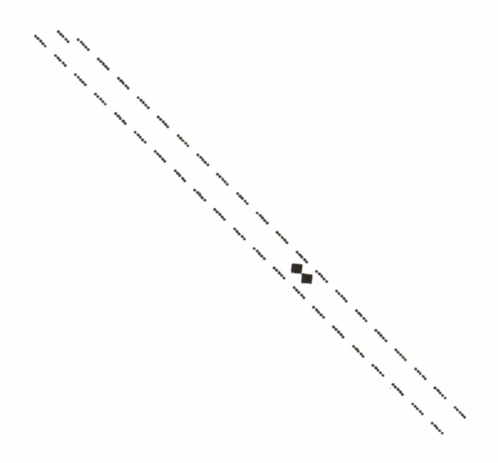


Fig. 3.5. La «hormiga» sigue un recorrido diagonal flanqueada por dos líneas discontinuas.

CHOQUE Y REBOTE

Hasta ahora hemos visto movimientos sobre distintas líneas rectas, es decir, en distintas direcciones, sin embargo el sentido de avance del carácter que se desplazaba era siempre el mismo. Ahora vamos a suponer que el carácter en movimiento, cuando llega a un extremo de la pantalla o a cualquier otro obstáculo que queramos, choca contra este límite e invierte el sentido del movimiento, es decir, el efecto que se produce es algo parecido a un rebote.

```
 10 REM *********************
 20 REM *  CHOQUE Y REBOTE   *
 30 REM *      SPECTRUM      *
 40 REM *********************
 50 INK 7: PAPER 1: BORDER 6: CLS
 60 FOR I=4 TO 21
 70 PRINT AT I,4;CHR$ 143;AT I,27;CHR$ 143
 80 NEXT I
 90 LET X=11: LET Y=16: LET A=1
100 PRINT AT X,Y;" "
110 LET Y=Y+A
120 PRINT  INK 2;AT X,Y;CHR$ 143
130 IF Y=26 THEN  LET A=-1
140 IF Y=5 THEN  LET A=1
150 PAUSE 5
160 GO TO 100
```

En el programa 3.9 «construimos» dos muros verticales, coincidiendo con las columnas 4 y 27 de la pantalla, mediante el bucle FOR-NEXT de las líneas 60-80. En la línea 90 asignamos unas coordenadas iniciales x e y en el centro de la pantalla, entre los dos muros. También asignamos a la variable A el valor 1. Esta variable A es muy importante ya que va a determinar el sentido de avance de una «pelota» roja que va a rebotar continuamente entre los dos muros. El valor de la columna donde debe imprimirse la «pelota» va variando en la línea 110. Aquí es donde podemos ver que si A vale 1, el número de columna aumenta y, por tanto, la pelota se desplazará hacia la derecha, mientras que si A toma el valor –1, el número de columna disminuye, haciendo que la pelota se mueva hacia la izquierda. Para variar los valores de A a –1 ó 1 están las condiciones de las líneas 130 y 140 que sirven para controlar cuando la pelota «choca» con alguno de los muros.

Cuando se produce uno de estos choques A toma el valor contrario al que tenía y por tanto se invierte el sentido del movimiento.

```
10 REM ***************************
20 REM *     CHOQUE Y REBOTE      *
30 REM *         COMMODORE        *
40 REM ***************************
50 PRINT CHR$(147)
60 FOR I=5 TO 24
70 LOCATE 5,I:PRINT CHR$(166)
80 LOCATE 34,I:PRINT CHR$(166)
90 NEXT I
100 X=5:Y=20:A=1
110 LOCATE Y,X:PRINT " "
120 Y=Y+A
130 LOCATE Y,X:PRINT CHR$(113)
140 IF Y=33 THEN A=-1
150 IF Y=6 THEN A=1
160 FOR I=1 TO 50:NEXT I
170 GOTO 110
```

Ver apéndice C.

El programa 3.10 tiene el mismo objetivo que el 3.9 pero es la versión para Commodore. Con unos pequeños cambios, reseñados en el apéndice A, sirve también para IBM y Amstrad.



Fig. 3.6. La «pelotita» rebota entre los dos muros.

Este programa es sólo una iniciación al efecto de choque y rebote, en forma de movimiento sencillo. Sin embargo, este tipo de movimientos pueden mejorarse mucho dando lugar a efectos más complejos. Todo esto lo veremos en capítulos posteriores.



MOVIMIENTO DE FIGURAS COMPLEJAS

Hasta aquí hemos visto el movimiento de figuras simples, compuestas por un solo carácter. Pero también es posible producir el movimiento de una figura compleja, compuesta por varios caracteres predefinidos o definidos por el usuario. La técnica para simular el desplazamiento es la misma que para un solo carácter por tanto vamos a pasar a ver directamente un programa para simular por ejemplo un ciempiés avanzando por la pantalla.

```
 10 REM ***********************
 20 REM * PASEO DEL CIEMPIES *
 30 REM *       SPECTRUM      *
 40 REM ***********************
 50 INK 7: PAPER 4: BORDER 6: CLS
 60 FOR I=1 TO 31 STEP 5
 70 PRINT AT 5,I;"*";AT 16,I;"*"
 80 NEXT I
 90 LET X=10: LET Y=0: LET A=1
100 INK 0
110 PRINT AT X-2,Y;"         "
120 PRINT AT X-1,Y;"  )))))  "
130 PRINT AT X+0,Y;" 0000000<"
140 PRINT AT X+1,Y;"  )))))  "
150 PRINT AT X+2,Y;"         "
160 LET Y=Y+1: LET X=X+A
170 LET A=-A
180 IF Y>23 THEN  PRINT AT 20,4;
    "!UY! QUE GOLPE MAS TONTO": GO TO 1000
190 PAUSE 10
200 GO TO 110
```

El ciempiés del programa 3.11 avanza haciendo eses. El avance hacia la derecha se produce por el incremento de la coordenada Y (línea 160), que hace que aumente el número de la columna donde debe realizarse la impresión del ciempiés en cada momento. El efecto de las eses se consigue haciendo que la coordenada X tome en cada impresión un valor distinto entre dos valores elegidos para las filas (línea 160), en este ejemplo las filas 10 y 11. Para ello utilizamos una variable A (línea 90) que va tomando de forma alternada los valores 1 y –1 (línea 170). De este modo, después de una impresión en la fila 10, se le suma a la coordenada X el valor 1, de modo que la siguiente impresión se realiza en la fila 11. A continuación la variable A cambia de signo tomando el valor –1, de modo que al sumarlo

al valor actual de X (11) vuelve a resultar el 10 como número de fila para la siguiente impresión. Este proceso se repite cada vez que avanza el ciempiés. Las líneas 110 y 150 imprimen los espacios en blanco necesarios para borrar las patas del ciempiés cada vez que hace una ese (un cambio de fila).

Por otra parte el bucle FOR-NEXT de las líneas 60-80 se encarga de «plantar» unas cuantas margaritas blancas en un verde «prado».

```
10 REM *****************************
20 REM *  EL PASEO DEL COCHECITO   *
30 REM *           IBM             *
40 REM *****************************
50 SCREEN 0:WIDTH 40:KEY OFF
60 COLOR 2:CLS
70 FOR I=1 TO 40
80 LOCATE 9,I:PRINT CHR$(178)
90 LOCATE 16,I:PRINT CHR$(178)
100 NEXT
110 X=12:Y=1:A=1
120 COLOR 4
130 LOCATE X-2,Y:PRINT SPC(6)
140 LOCATE X-1,Y:PRINT "  ";CHR$(201);
    CHR$(203);CHR$(187);" "
150 LOCATE X,Y:PRINT " ";CHR$(219);
    CHR$(219);CHR$(219);CHR$(219);CHR$(219)
160 LOCATE X+1,Y:PRINT "  O O "
170 LOCATE X+2,Y:PRINT SPC(6)
180 Y=Y+1:X=X+A:A=-A
190 IF Y>35 THEN LOCATE 22,8:PRINT
    "¡UY! QUE GOLPE MAS TONTO":COLOR 7:END
200 FOR I=1 TO 100:NEXT
210 GOTO 130
```

Esta versión sirve para el MSX.

El programa 3.12 presenta un cochecito similar al ciempiés del programa 3.11, sólo que ahora se pasea por la pantalla del IBM; el programa sería análogo en el Amstrad y el Commodore.

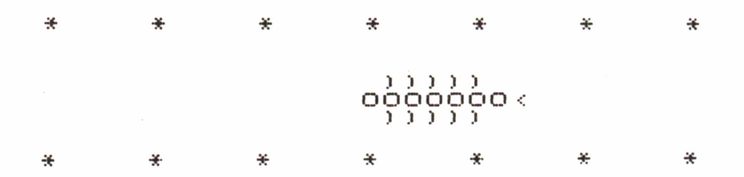
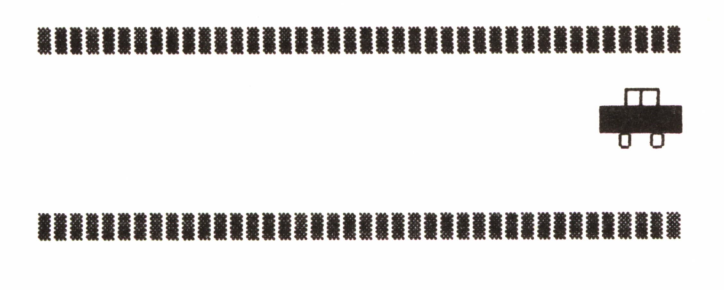


Fig. 3.7. El «ciempiés mareado» va haciendo eses por la pantalla.

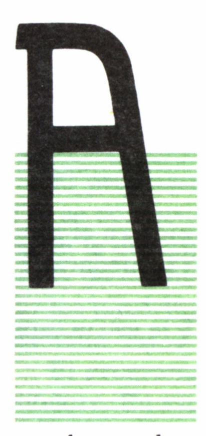
Bueno, esto empieza a ser divertido pero, de momento, es el ordenador el encargado de controlar los movimientos de los personajes. Sería interesante que nosotros pudiéramos mover una figura por toda la pantalla, pero eso forma parte ya de otro capítulo.



```
          ¡UY! QUE GOLPE MAS TONTO
Ok
```

Fig. 3.8. El cochecito avanza por la calle haciendo eses hasta que choca al llegar al final de la pantalla.

CONTROLANDO EL MOVIMIENTO 4





INTRODUCCION

A lo largo del capítulo anterior hemos visto cómo el ordenador producía el movimiento de todo tipo de figuras en cualquier dirección. Sin embargo, de esta forma el usuario permanece pasivo una vez que ha realizado el programa. En el presente capítulo vamos a descubrir los secretos que hacen posible el control del movimiento por el usuario, ya que es fundamental de cara al diseño de cualquier videojuego, por sencillo que sea. En más de una ocasión hemos tenido oportunidad de ver algún videojuego en el que el protagonista de la aventura tiene que evitar a todos los enemigos que le asedian. La idea principal se basa en unas cuantas figuras (los enemigos) cuyo movimiento produce el ordenador y el personaje principal que se mueve a nuestra voluntad.



LA FUNCION INKEY$

Antes de entrar en el tema del control del movimiento propiamente dicho, vamos a estudiar una función fundamental para la comprensión del resto del capítulo.

Esta función es INKEY$, aunque en algunos ordenadores como el Commodore se denomina también GET.

A grandes rasgos, la misión de INKEY$ es «leer» el teclado en el momento de su ejecución. Si en esta lectura encuentra alguna tecla pulsada, el carácter o código de dicha tecla queda almacenado en memoria, pero si en el momento de la lectura no pulsamos ninguna tecla, la ejecución del programa continúa, almacenándose en la parte de la memoria correspondiendo a INKEY$ la cadena vacía. INKEY$ es una función alfanumérica, por tanto, aunque pulsemos una tecla numérica, ésta se almacenará en for-

ma de cadena. Por otra parte, la función INKEY$ no «espera» a que pulsemos una tecla, por lo que es corriente utilizar la estructura siguiente:

```
10 IF INKEY$="" THEN GOTO 10
```

De este modo se produce una pausa en la ejecución del programa, que no se romperá hasta que no pulsemos una tecla cualquiera.

Otra estructura bastante común es la siguiente:

```
10 A$=INKEY$ : IF A$= "" THEN GOTO 10
```

Al igual que antes, esta línea produciría una pausa indefinida en el programa, pero en este caso el contenido de la función INKEY$ quedará almacenado en la variable alfanumérica A$.

En los ordenadores que utilizan GET en lugar de INKEY$ la estructura para generar la pausa indefinida es la siguiente:

```
10 GET A$ : IF A$= "" THEN GOTO 10
```

Por último diremos que la función INKEY$ va a ser la que nos permita controlar el movimiento de una figura cualquiera por la pantalla del ordenador, de modo que ¡adelante!



PRIMERAS IDEAS PARA EL CONTROL DEL MOVIMIENTO

Vamos a comenzar por un problema muy simple: controlar el movimiento de un carácter de izquierda a derecha de la pantalla. Para conseguirlo vamos a seguir la idea siguiente: imprimimos un carácter cualquiera en una posición determinada de la pantalla; a continuación generamos una pausa indefinida que sólo se romperá cuando pulsemos una tecla determinada (en este caso la más apropiada es la flecha hacia la derecha →). Cada vez que pulsemos dicha tecla el carácter en cuestión avanzará una posición hacia la derecha.

```
 10 REM *****************
 20 REM * PRIMER CONTROL *
 30 REM * DEL MOVIMIENTO *
 40 REM *    SPECTRUM    *
 50 REM *****************
 60 LET X=11: LET Y=0
 70 PRINT AT X,Y;CHR$ (137)
 80 PAUSE 10
 90 LET A$=INKEY$
100 IF A$="" THEN  GO TO 90
110 IF CODE A$<>56 THEN  GO TO 90
120 LET Y=Y+1
130 IF Y>30 THEN  LET Y=0
140 GO TO 70
```

El programa 4.1 lleva a cabo estas ideas en el Spectrum. En la línea 110 se comprueba si el código de la tecla pulsada corresponde a la flecha hacia la derecha (el 56 es el código del 8 que es la tecla donde se encuentra la mencionada flecha). La condición de la línea 130 permite que cuando los cuadritos que estamos moviendo lleguen al extremo derecho de la pantalla, éstos vuelvan a aparecer por el extremo izquierdo.

El programa 4.2 consigue el mismo efecto que el 4.1 pero en el Commodore. En este caso la tecla que hay que pulsar para que la figura avance es la D (código 68).

```
10 REM ********************
20 REM *  PRIMER CONTROL  *
30 REM *  DEL MOVIMIENTO  *
40 REM *     COMMODORE    *
50 REM ********************
60 PRINT CHR$(147)
70 X=12:Y=0
80 LOCATE Y,X:PRINT " ";CHR$(113)
90 FOR I=1 TO 100:NEXT
100 GET A$
110 IF A$="" THEN GOTO 100
120 IF ASC(A$)<>68 THEN LOCATE Y,X:
    PRINT " ":Y=0
130 Y=Y+1
140 IF Y>38 THEN Y=0
150 GOTO 80
```

Ver apéndice C.

Sin embargo, normalmente el personaje en movimiento no suele superar los límites de la pantalla, es decir, sería más interesante que el carác-

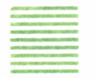
ter que estamos moviendo, al llegar al extremo de la pantalla, chocara con este límite, es decir, que no volviera a aparecer por la izquierda para seguir avanzando. Para conseguir este efecto no hay más que introducir una mínima modificación. En el programa para Spectrum sólo hay que sustituir la línea 130 por:

```
130 IF Y>30 THEN LET Y=30
```

Mientras que en el Commodore habría que sustituir la línea 140 por:

```
140 IF Y>38 THEN Y=38
```

De este modo cuando lleguemos al extremo de la pantalla «chocaremos» y no podremos seguir avanzando.



CONTROL GENERAL DEL MOVIMIENTO

Ahora que ya sabemos la idea a seguir para controlar el movimiento en un solo sentido, el siguiente paso es desarrollar un programa que nos permite «desplazarnos» por la pantalla en cualquier dirección. Para ello vamos a seguir el mismo esquema que en el apartado anterior, aunque en esta ocasión vamos a utilizar cuatro teclas para el control del movimiento según cuatro sentidos: izquierda, derecha, arriba y abajo. Normalmente se utilizan las cuatro flechas de movimiento del cursor.

```
 10 REM ***********************
 20 REM * MOVIMIENTO GENERAL *
 30 REM *       SPECTRUM      *
 40 REM ***********************
 50 LET X=11: LET Y=16
 60 PRINT AT X-1,Y+1;" ";AT X,Y;" * ";
    AT X+1,Y+1;" "
 70 PAUSE 10
 80 LET A$=INKEY$
 90 IF A$="" THEN GO TO 80
100 IF CODE A$<53 OR CODE A$>56 THEN
```

```
    GO TO 80
110 GO SUB VAL A$*100
120 GO TO 60
500 REM * MOVIMIENTO HACIA *
510 REM *   LA IZQUIERDA    *
520 LET Y=Y-1
530 IF Y<0 THEN  LET Y=0
540 RETURN
600 REM * MOVIMIENTO HACIA *
610 REM *       ABAJO       *
620 LET X=X+1
630 IF X>20 THEN  LET X=20
640 RETURN
700 REM * MOVIMIENTO HACIA *
710 REM *       ARRIBA      *
720 LET X=X-1
730 IF X<1 THEN  LET X=1
740 RETURN
800 REM * MOVIMIENTO HACIA *
810 REM *    LA DERECHA     *
820 LET Y=Y+1
830 IF Y>30 THEN  LET Y=30
840 RETURN
```

El programa 4.3 nos permite mover un asterisco por toda la pantalla del ordenador, siguiendo 4 sentidos distintos. Para ello utilizamos las flechas que están en las teclas 5, 6, 7 y 8, cuyos códigos ASCII varían entre 53 y 56. En la línea 100 ponemos una condición para que el asterisco no se mueva si pulsamos cualquier tecla distinta de estas cuatro. Cada movimiento está desarrollado en una subrutina independiente aunque la estructura genérica es similar en las cuatro, ya que se trata de avanzar o retroceder una posición en una dirección determinada, según la flecha que hayamos pulsado (líneas 520, 620, 720 y 820). Además en cada subrutina controlamos que el asterisco no pueda salirse de la pantalla, haciendo que cuando llegue al borde choque, es decir, se mantenga la posición en la que está. Esto lo conseguimos con las condiciones de las líneas 530, 630, 730 y 830.

El programa 4.4 es análogo al 4.3 en cuanto a resultados, aunque es la versión para Amstrad y, con unas sencillas modificaciones, para IBM y Commodore. Las teclas utilizadas para el control son las flechas de movimiento del cursor.

```
10 REM *************************
20 REM *  MOVIMIENTO GENERAL   *
30 REM *        AMSTRAD        *
40 REM *************************
```

```
50 MODE 1 :CLS
60 X=13:Y=20
70 LOCATE Y+1,X-1:PRINT " "
80 LOCATE Y,X:PRINT " ";CHR$(248);" "
90 LOCATE Y+1,X+1:PRINT " "
100 FOR I=1 TO 100:NEXT
110 A$=INKEY$
120 IF A$="" THEN GOTO 110
130 IF ASC(A$)<240 OR ASC(A$)>243
    THEN GOTO 110
140 A=ASC(A$)-239
150 ON A GOSUB 500,600,700,800
160 GOTO 70
500 REM * MOVIMIENTO HACIA ARRIBA *
510 X=X-1
520 IF X<2 THEN X=2
530 GOTO 70
600 REM * MOVIMIENTO HACIA ABAJO *
610 X=X+1
620 IF X>22 THEN X=22
630 RETURN
700 REM * MOVIMIENTO HACIA LA IZQUIERDA *
710 Y=Y-1
720 IF Y<1 THEN Y=1
730 RETURN
800 REM * MOVIMIENTO HACIA LA DERECHA *
810 Y=Y+1
820 IF Y>39 THEN Y=39
830 RETURN
```

Para MSX e IBM, ver apéndice B.

Muy bien, ya dominamos por completo el movimiento de un solo carácter por la pantalla, ahora tenemos que conseguir movimientos de figuras más complejas.



CONTROL DEL MOVIMIENTO DE FIGURAS COMPLEJAS

Si queremos mover una figura compuesta por varios caracteres por la pantalla no tenemos más que seguir la misma metodología vista hasta ahora, teniendo cuidado de imprimir cada «trozo» de la figura en el sitio correcto.

Vamos a dibujar un invasor galáctico utilizando ocho caracteres gráficos y a continuación podremos moverlo por toda la pantalla.

```
 10 REM **********************
 20 REM *MOVIENDO AL MARCIANO*
 30 REM *      SPECTRUM      *
 40 REM **********************
 50 INK 6: PAPER 1: BORDER 5
 60 CLS
 70 LET X=11: LET Y=14
 80 PRINT AT X-1,Y+1;CHR$ 137;CHR$ 134
 90 PRINT AT X,Y;CHR$ 139;CHR$ 143;
    CHR$ 143;CHR$ 135
100 PRINT AT X+1,Y;CHR$ 142;"  ";CHR$ 141
110 PAUSE 5
120 LET A$=INKEY$
130 IF A$="" THEN  GO TO 120
140 IF CODE A$<53 OR CODE A$>56 THEN
    GO TO 120
150 PRINT AT X-1,Y+1;"  "
160 PRINT AT X,Y;"    "
170 PRINT AT X+1,Y;"    "
180 GO SUB VAL (A$)*100
190 GO TO 80
500 REM * IZQUIERDA *
510 LET Y=Y-1
520 IF Y<0 THEN  LET Y=0
530 RETURN
600 REM * ABAJO *
610 LET X=X+1
620 IF X>20 THEN  LET X=20
630 RETURN
700 REM * ARRIBA *
710 LET X=X-1
720 IF X<1 THEN  LET X=1
730 RETURN
800 REM * DERECHA *
810 LET Y=Y+1
820 IF Y>28 THEN  LET Y=28
830 RETURN
```

El programa 4.5 nos permite mover al invasor galáctico por la pantalla del Spectrum. Los caracteres gráficos utilizados están representados en el listado por sus códigos ASCII asociados (utilizando la función CHR$). El «marciano» obtenido está representado en la figura 4.1.



Fig. 4.1. Diseño de invasor galáctico.

Por último, incluimos el programa 4.6, que es una versión del programa 4.5 para IBM. La figura diseñada es distinta pero el objetivo es el mismo. Las teclas utilizadas para el control del movimiento son las de los números 5 (arriba), 6 (abajo), 7 (izquierda) y 8 (derecha).

```
10 REM ************************
20 REM * MOVIENDO AL MARCIANO *
30 REM *         IBM          *
40 REM ************************
50 SCREEN 0:WIDTH 40:KEY OFF
60 COLOR 10,4,12:CLS
70 X=12:Y=18
80 LOCATE X-1,Y:PRINT CHR$(222);
   CHR$(223);CHR$(221)
90 LOCATE X,Y:PRINT CHR$(220);
   CHR$(219);CHR$(220)
100 LOCATE X+1,Y:PRINT CHR$(221);
    CHR$(223);CHR$(222)
110 FOR I=1 TO 100:NEXT
120 A$=INKEY$
130 IF A$="" THEN GOTO 120
140 IF ASC(A$)<53 OR ASC(A$)>56 THEN GOTO 120
150 LOCATE X-1,Y:PRINT "   "
160 LOCATE X,Y:PRINT "   "
170 LOCATE X+1,Y:PRINT "   "
180 ON ASC(A$)-52 GOSUB 500,600,700,800
190 GOTO 80
500 REM * ARRIBA *
510 X=X-1
520 IF X<2 THEN X=2
530 RETURN
600 REM * ABAJO *
610 X=X+1
620 IF X>21 THEN X=21
630 RETURN
700 REM * DERECHA *
710 Y=Y+1
720 IF Y>38 THEN Y=38
730 RETURN
800 REM * IZQUIERDA *
810 Y=Y-1
820 IF Y<1 THEN Y=1
830 RETURN
```

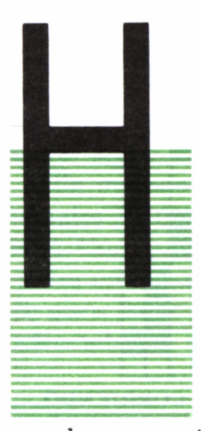
Esta versión sirve para el MSX.

Estos dos últimos programas no son más que una aplicación de todo lo que hemos visto hasta el momento, por tanto no creemos necesario dar una explicación sobre su planteamiento y funcionamiento, puesto que es similar al de los demás ejemplos vistos a lo largo de los capítulos 3 y 4.

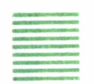
ANIMACION DE PERSONAJES 5

INTRODUCCION





HASTA ahora hemos visto todo tipo de figuras, sencillas y complejas, trasladándose de un punto a otro de la pantalla; sin embargo, estos «paseos» son como «a saltos», ya que la figura que imprimimos para simular el movimiento es siempre la misma. Sería interesante conseguir que un personaje, a la vez que se traslada de un sitio a otro, moviera las piernas, los brazos o incluso la cabeza. Esto es posible siempre que seamos capaces de seguir un orden correcto a la hora de imprimir en pantalla cada «trozo» de personaje.



LAS POSTURAS DEL PERSONAJE

En primer lugar vamos a elegir el tipo de personaje que queremos y las diferentes posturas que podría adoptar. Para el desarrollo del capítulo hemos elegido como ejemplo un hombrecito compuesto por tres caracteres en vertical: la cabeza, el tronco con los brazos y las piernas. Por otra parte, vamos a hacer que nuestro hombre pueda correr hacia la izquierda y hacia la derecha, o quedarse parado mirando a izquierda y derecha. Esto implica un número muy variado de posturas a adoptar, ya que, en cada caso, tanto la cabeza como los brazos y las piernas adoptarán distintas posiciones. La figura 5.1 muestra todas las posturas posibles del hombrecito.

Evidentemente todos estos caracteres no están definidos en el teclado, por tanto el primer paso será su definición aplicando los métodos vistos en el capítulo 2.

Aquí vamos a desarrollar el programa completo en dos ordenadores distintos: el Spectrum y el Amstrad, que son los que presentan la posibilidad de definición de caracteres en BASIC, de una forma bastante sencilla. De

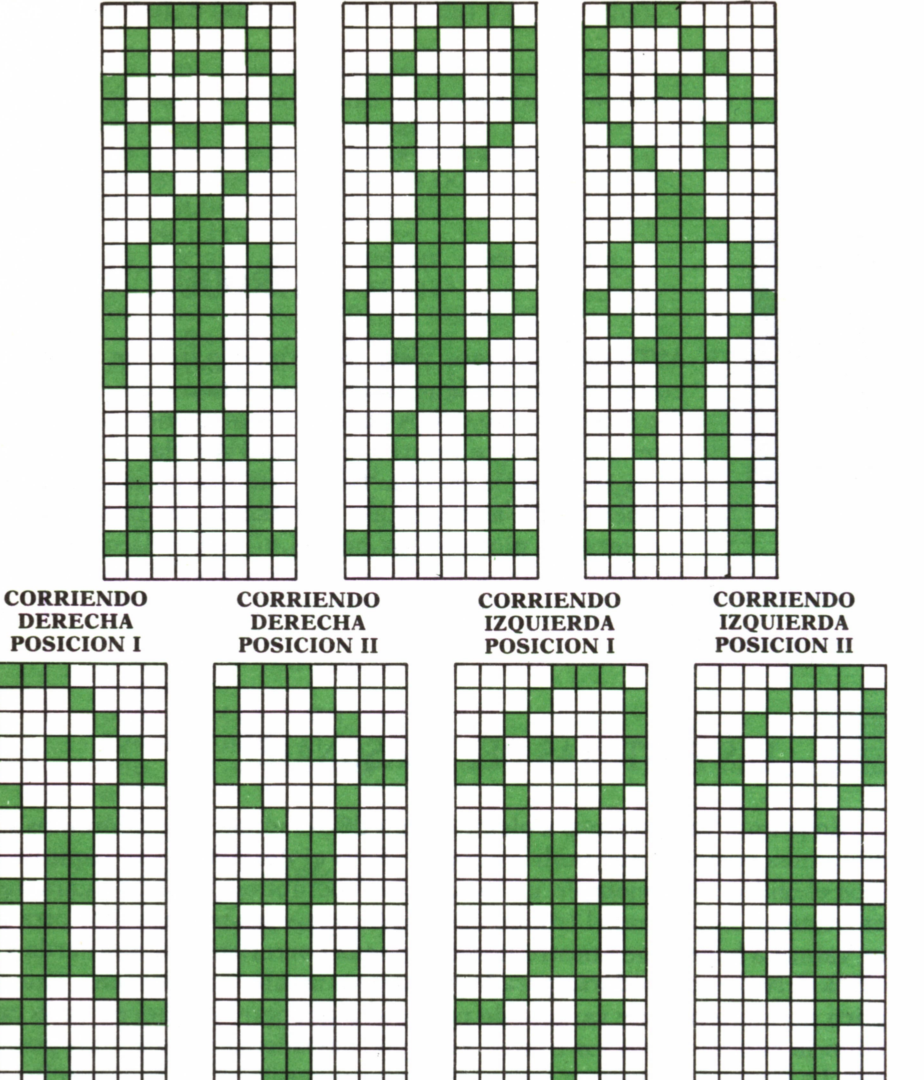


Fig. 5.1. Esquemas de las distintas posiciones del hombrecito, definidas cada una de ellas en tres caracteres.

cualquier modo, hay que tener en cuenta que hemos elegido el caso más complejo, ya que podemos conseguir la animación de personajes compuestos a base de caracteres ya definidos.



EL PERSONAJE DEL SPECTRUM

Si tenemos que definir 14 caracteres distintos (3 cabezas, 6 troncos y 5 piernas) no podemos hacer 14 por 8 POKE sobre otras tantas direcciones de memoria para introducir todos los binarios necesarios para la definición de todas las posturas de nuestro hombre. Esto sería demasiado largo y engorroso (112 líneas de programa), aparte de que sería muy fácil cometer algún error.

Un método mucho más corto es transformar todos los binarios en decimales (podemos consultar la tabla de conversión incluida al final del libro). Una vez hecha esta operación almacenamos todos los números decimales en líneas DATA y luego con un sencillo bucle se realiza la lectura de estos datos y se almacena cada uno de ellos en la posición de memoria adecuada.

```
 10 REM ******************
 20 REM * DEFINICION DEL *
 30 REM *   HOMBRECITO   *
 40 REM *    SPECTRUM    *
 50 REM ******************
 60 DATA 60,66,90,129,165,90,66,36
 70 DATA 14,17,33,89,193,34,36,24
 80 DATA 112,136,132,154,131,68,36,24
 90 DATA 24,60,90,90,153,153,153,153
100 DATA 24,60,90,90,153,90,60,24
110 DATA 24,27,13,13,29,36,70,4
120 DATA 24,30,9,77,46,20,6,4
130 DATA 24,216,176,176,184,36,98,32
140 DATA 24,120,144,178,116,40,96,32
150 DATA 24,36,36,66,66,66,195,0
160 DATA 12,8,27,42,68,64,192,0
170 DATA 4,12,20,34,34,17,35,0
180 DATA 48,16,216,84,34,2,3,0
190 DATA 32,48,40,68,68,136,196,0
200 FOR I=0 TO 111
210 READ N
220 POKE USR "A"+I,N
230 NEXT I
```

Para MSX, ver apéndice B.

En el programa 5.1 hemos utilizado teclas correlativas para definir los distintos caracteres (de la A a la N).

La correspondencia entre cada tecla y el carácter definido en ella es la siguiente:

TECLA	CODIGO	CARACTER DEFINIDO
A	144	Cabeza mirando al frente
B	145	Cabeza mirando a la izquierda
C	146	Cabeza mirando a la derecha
D	147	Cuerpo brazos extendidos
E	148	Cuerpo brazos en jarras
F	149	Cuerpo corriendo izquierda posición I
G	150	Cuerpo corriendo izquierda posición II
H	151	Cuerpo corriendo derecha posición I
I	152	Cuerpo corriendo derecha posición II
J	153	Piernas parado
K	154	Piernas corriendo izquierda posición I
L	155	Piernas corriendo izquierda posición II
M	156	Piernas corriendo derecha posición I
N	157	Piernas corriendo derecha posición II

Si queremos imprimirlos en pantalla no tenemos más que pulsar cada una de estas teclas en modo gráfico (CAPS SHIFT y GRAPHICS), aunque resulta mucho más elegante ejecutar un sencillo programa como el 5.2, que utiliza los códigos ASCII.

```
1000 REM *****************
1010 REM *LOS "TROZOS" DEL*
1020 REM *   HOMBRECITO   *
1030 REM *    SPECTRUM    *
1040 REM *****************
1050 FOR I=144 TO 157
1060 PRINT CHR$ (I);" ";
1070 NEXT I
```

Conviene recordar que los nuevos caracteres definidos permanecen almacenados en memoria aun después de hacer NEW y que las únicas formas de eliminarlos son desenchufando el ordenador o redefiniéndolos otra vez. Por tanto una vez ejecutado el programa 5.1 podríamos borrarlo de la memoria e introducir el 5.2 pero de todos modos resulta más claro si están los dos unidos en un solo programa.

Al ejecutar el programa 5.2 obtendríamos en pantalla todos los caracteres que muestra la figura 5.2.

Fig. 5.2. Conjunto de caracteres definidos para las distintas posturas del hombrecito con los programas 5.1 ó 5.3.

Bueno, ya sólo tenemos que ordenarlos convenientemente para que nuestro hombrecito comience a pasear, pero esto lo veremos un poco más adelante.



EL PERSONAJE DEL AMSTRAD

El procedimiento a seguir en este ordenador es análogo al del Spectrum sólo que en este caso utilizaremos las instrucciones **SYMBOL** y **SYMBOL AFTER** para redefinir caracteres. Las líneas **DATA**, al igual que antes almacenan los números decimales que hay que introducir en cada byte y que son los mismos que en el Spectrum, por tanto estas líneas no las hemos incluido en el programa 5.3. Por otra parte el bucle controla ahora los códigos que vamos a redefinir (en lugar de las direcciones de memoria). En cada vuelta del bucle se realizan ocho lecturas y se introducen en cada código los ocho números necesarios para definir el nuevo carácter.

```
10 REM ****************************
20 REM *DEFINICION DEL HOMBRECITO*
30 REM *          AMSTRAD         *
40 REM ****************************
50 MODE 1:CLS
200 SYMBOL AFTER 176
210 FOR I=176 TO 189
220 READ A,B,C,D,E,F,G,H
230 SYMBOL I,A,B,C,D,E,F,G,H
240 NEXT
```

Hemos utilizado los códigos entre 176 y 189 en los que antes había almacenadas unas cuantas letras griegas. Si queremos visualizar los caracteres como en la figura 5.2 no tenemos más que realizar un programa análogo al 5.2 sólo que variando los límites del bucle, es decir, sustituyendo la línea 1050 por:

```
1050 FOR I=176 TO 189
```

Los caracteres redefinidos en cada código siguen el mismo orden que en el Spectrum.

Bueno, pues ya ha llegado el momento de ver todos los movimientos que puede realizar nuestro hombre.



LOS PASEOS DEL PERSONAJE

Ya podemos pasear a nuestro hombrecito por la pantalla. Veamos qué movimientos es capaz de hacer. En primer lugar cuando esté parado en algún punto de la pantalla, moverá la cabeza a izquierda, frente y derecha alternativamente, a la vez que mueve los brazos desde la posición de jarras hasta extenderlos.

Por otra parte si pulsamos la tecla de la flecha hacia la derecha el hombrecito empezará a correr en esa dirección, mientras que si pulsamos la tecla de la flecha hacia la izquierda correrá en sentido contrario.

```
300 REM ************************
310 REM *PASEOS DEL HOMBRECITO*
320 REM *       SPECTRUM       *
330 REM ************************
340 LET X=10: LET Y=16
350 GO SUB 500
360 PRINT AT X,Y;CHR$ 144;AT X+1,Y;
    CHR$ 148;AT X+2,Y;CHR$ 153
370 PAUSE 10
380 PRINT AT X,Y;CHR$ 145;AT X+1,Y;CHR$ 147
390 PAUSE 10
400 PRINT AT X,Y;CHR$ 144;AT X+1,Y;CHR$ 148
410 PAUSE 10
420 PRINT AT X,Y;CHR$ 146;AT X+1,Y;CHR$ 147
430 PAUSE 10
440 GO TO 350
500 REM * MOVIMIENTO *
510 LET A$=INKEY$
520 IF CODE A$=53 THEN  GO SUB 600: GO TO 510
530 IF CODE A$=56 THEN  GO SUB 700: GO TO 510
540 RETURN
600 REM * IZQUIERDA *
610 PRINT AT X,Y;" ";AT X+1,Y;" ";AT X+2,Y;" "
620 LET Y=Y-1
630 IF Y<0 THEN  LET Y=31
640 PRINT AT X,Y;CHR$ 145;AT X+1,Y;
    CHR$ 149;AT X+2,Y;CHR$ 154
```

```
650 PAUSE 5
660 PRINT AT X+1,Y;CHR$ 150;AT X+2,Y;CHR$ 155
670 PAUSE 5
680 RETURN
700 REM * DERECHA *
710 PRINT AT X,Y;" ";AT X+1,Y;" ";AT X+2,Y;" "
720 LET Y=Y+1
730 IF Y>31 THEN  LET Y=0
740 PRINT AT X,Y;CHR$ 146;AT X+1,Y;
    CHR$ 151;AT X+2,Y;CHR$ 156
750 PAUSE 5
760 PRINT AT X+1,Y;CHR$ 152;AT X+2,Y;CHR$ 157
770 PAUSE 5
780 RETURN
```

El único «misterio» del programa 5.4 consiste en imprimir los caracteres adecuados a la situación en la que se encuentre el hombrecito en cada momento. Evidentemente, dichos caracteres son los definidos en el programa 5.1 por tanto conviene unir ambos programas para que el funcionamiento sea correcto.

```
300 REM ************************
310 REM *PASEOS DEL HOMBRECITO*
320 REM *       AMSTRAD        *
330 REM ************************
340 X=13:Y=20
350 GOSUB 500
360 LOCATE Y,X:PRINT CHR$(144)
370 LOCATE Y,X+1:PRINT CHR$(143)
380 LOCATE Y,X+2:PRINT CHR$(153)
390 FOR I=1 TO 20:NEXT
400 LOCATE Y,X:PRINT CHR$(145)
410 LOCATE Y,X+1:PRINT CHR$(147)
420 FOR I=1 TO 20:NEXT
430 LOCATE Y,X:PRINT CHR$(144)
440 LOCATE Y,X+1:PRINT CHR$(148)
450 FOR I=1 TO 20:NEXT
460 LOCATE Y,X:PRINT CHR$(146)
470 LOCATE Y,X+1:PRINT CHR$(147)
480 FOR I=1 TO 20:NEXT
490 GOTO 350
500 REM * MOVIMIENTO *
510 A$=INKEY$
520 IF ASC(A$)=242 THEN GOSUB 600:GOTO 510
530 IF ASC(A$)=243 THEN GOSUB 800:GOTO 510
```

```
540 RETURN
600 REM * IZQUIERDA *
610 LOCATE Y,X:PRINT " "
620 LOCATE Y,X+1:PRINT " "
630 LOCATE Y,X+2:PRINT " "
640 Y=Y-1
650 IF Y<1 THEN Y=40
660 LOCATE Y,X:PRINT CHR$(145)
670 LOCATE Y,X+1:PRINT CHR$(149)
680 LOCATE Y,X+2:PRINT CHR$(154)
690 FOR I=1 TO 20:NEXT
700 LOCATE Y,X+1:PRINT CHR$(150)
710 LOCATE Y,X+2:PRINT CHR$(155)
720 FOR I=1 TO 20:NEXT
730 RETURN
800 REM * DERECHA *
810 LOCATE Y,X:PRINT " "
820 LOCATE Y,X+1:PRINT " "
830 LOCATE Y,X+2:PRINT " "
840 Y=Y+1
850 IF Y>40 THEN Y=1
860 LOCATE Y,X:PRINT CHR$(146)
870 LOCATE Y,X+1:PRINT CHR$(151)
880 LOCATE Y,X+2:PRINT CHR$(156)
890 FOR I=1 TO 20:NEXT
900 LOCATE Y,X+1:PRINT CHR$(152)
910 LOCATE Y,X+2:PRINT CHR$(157)
920 FOR I=1 TO 20:NEXT
930 RETURN
```

Para MSX, ver apéndice B.

El programa 5.5 tiene el mismo objetivo que el anterior, pero es la versión para Amstrad. Al igual que el Spectrum, conviene unirlo a la rutina de definición de los caracteres (programa 5.3).

Al ejecutar cualquiera de estos dos programas podemos observar que cuando el hombrecito llega a un extremo de la pantalla desaparece y vuelve a aparecer por el extremo opuesto.

Por último, en la figura 5.3 podemos ver las diferentes posturas que adopta nuestro hombrecito en sus paseos por la pantalla.

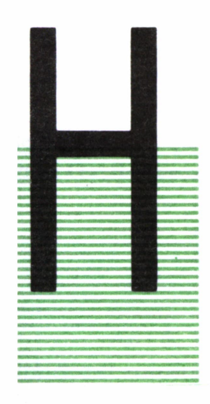


Fig. 5.3. Estas son las diferentes posturas que adopta el hombrecito en la pantalla del ordenador.

JUGANDO A DESPISTAR 6



INTRODUCCION



HASTA ahora todos los movimientos que hemos visto se podían predecir, es decir, en todo momento sabíamos cuál iba a ser la dirección que iba a tomar el personaje en su desplazamiento, bien fuera porque éramos nosotros mismos los que controlábamos su movimiento o porque el ordenador estaba programado para moverlo en una determinada dirección. Sin embargo, todos hemos tenido, en alguna ocasión, la oportunidad de ver algún videojuego en el que no sabíamos realmente los movimientos que iban a realizar cada uno de los personajes. Evidentemente, si supiéramos de antemano por qué parte de la pantalla van a atacar, por ejemplo, los marcianos, sería muy fácil que la nave terrícola los eliminara.

Pues bien, a lo largo de este capítulo vamos a ver cómo podemos programar movimientos no predecibles, que son, quizá, los más interesantes.



LAS FUNCIONES RND Y RANDOMIZE

Vamos a empezar por el estudio de una función fundamental para el desarrollo del resto del capítulo: RND. La función RND genera números al azar mayores o iguales que 0 y menores que 1. Sin embargo, es difícil imaginar que un ordenador, tan sometido a la matemática y la lógica, sea capaz de jugar con el azar. En sentido estricto, los números generados por RND no son realmente aleatorios sino que son el resultado de una serie de complejos cálculos que el ordenador realiza con tal rapidez que, desde un punto de vista práctico, resultan impredecibles. Por este motivo, los números generados mediante la función RND se llaman seudoaleatorios.

Por otra parte, aunque los números que podemos obtener están comprendidos entre 0 y 1, podríamos variar este intervalo valiéndonos de unos

sencillos artificios. Por ejemplo, si quisiéramos simular el lanzamiento de un dado por el ordenador, se trataría de generar un número entero al azar comprendido entre 1 y 6. Las transformaciones a realizar serían las siguientes:

0=< RND < 1	Números entre 0 y 1 (excluido)
0=< RND*6<6	Números entre 0 y 6 (excluido)
0=< INT(RND*6)<6	Números enteros entre 0 y 6 (excluido)
1=<INT(RND*6)+1<7	Números enteros entre 1 y 7 (excluido)

Por tanto, la función INT(RND*6)+1 simularía de una forma muy convincente el lanzamiento de un dado.

Hay que señalar que la función RND tiene argumento opcional en el Amstrad y el IBM, mientras que en el Commodore es obligatorio. RND con argumento tiene el formato siguiente:

RND (n)

donde *n* es un número que normalmente será positivo, aunque también puede ser 0 o negativo, teniendo en estos dos últimos casos funciones diferentes.

Sin embargo, RND produce la misma serie de resultados cada vez que se enciende el ordenador, lo cual evidencia su carácter seudoaleatorio. Para conseguir una sensación de casualidad total en BASIC existe la función RANDOMIZE (abreviadamente RAND). Esta función actúa como puntero encargado de indicar a RND dónde debe iniciar la serie de números seudoaleatorios. Con RANDOMIZE 0 se consigue la mayor casualidad posible en el Spectrum, ya que la serie de números generada por RND se iniciará dependiendo del tiempo que lleve encendido el ordenador. En Amstrad se consigue un resultado equivalente con RANDOMIZE TIME y en IBM con RANDOMIZE TIMER. El Commodore no dispone de RANDOMIZE.

```
10 REM ************
20 REM *   DADO   *
30 REM * SPECTRUM *
40 REM ************
50 RANDOMIZE 0: CLS
60 LET NV=INT (RND*12)+1
70 FOR I=1 TO NV
```

```
  80 LET D=INT (RND*6)+1
  90 CLS
 100 GO SUB D*1000
 110 PAUSE 25
 120 NEXT I
 130 PRINT AT 21,8;"OTRA TIRADA? (S/N)"
 140 LET A$=INKEY$: IF A$="" THEN  GO TO 130
 150 IF A$="S" OR A$="s" THEN  GO TO 50
 160 IF A$<>"N" AND A$<>"n" THEN  GO TO 130
 170 GO TO 9000
1000 REM * UNO *
1010 PRINT AT 11,16;CHR$ (143)
1020 RETURN
2000 REM * DOS *
2010 PRINT AT 9,14;CHR$ (143);AT 13,18;
     CHR$ (143)
2020 RETURN
3000 REM * TRES *
3010 PRINT AT 9,14;CHR$ (143);AT 11,16;
     CHR$ (143);AT 13,18;CHR$ (143)
3020 RETURN
4000 REM * CUATRO *
4010 PRINT AT 9,14;CHR$ (143);"   ";CHR$ (143)
     ;AT 13,14;CHR$ (143);"   ";CHR$ (143)
4020 RETURN
5000 REM * CINCO *
5010 PRINT AT 9,14;CHR$ (143);"   ";
     CHR$ (143);AT 11,16;CHR$ (143);AT 13,14;
     CHR$ (143);"   ";CHR$ (143)
5020 RETURN
6000 REM * SEIS *
6010 PRINT AT 9,14;CHR$ (143);"   ";
     CHR$ (143);AT 11,14;CHR$ (143);"   ";CHR$
     (143);AT 13,14;CHR$ (143);"   ";CHR$ (143)
6020 RETURN
```

El programa 6.1 es un ejemplo de la utilización de **RND** y **RANDOMIZE** para simular el lanzamiento de un dado en el Spectrum. En la línea 60 se determina al azar el número de vueltas que va a dar el dado antes de pararse con el resultado definitivo. El bucle **FOR-NEXT** de las líneas 70 a 110 se encarga de generar, a cada vuelta del dado, un número entre 1 y 6 para, a continuación ir a la subrutina correspondiente al número obtenido e imprimir en pantalla la cara del dado que haya salido. Cada cara del dado se imprime en pantalla en una subrutina independiente (de la 1000 a la 6000). El resultado de una posible ejecución está representado en la figura 6.1.



Fig. 6.1. Esta es una de las caras del lado que puede salir en pantalla al ejecutar los programas 6.1 ó 6.2.

El programa 6.2 simula el lanzamiento de un dado, igual que el programa anterior, pero esta vez utilizando un Commodore.

```
10 REM *************
20 REM *    DADO    *
30 REM * COMMODORE *
40 REM *************
50 PRINT CHR$(147)
60 NV=INT(RND(1)*12)+1
70 FOR I=1 TO NV
80 D=INT(RND(1)*6)+1
90 PRINT CHR$(147)
100 ON D GOSUB 1000,2000,3000,4000,5000,6000
110 FOR J=1 TO 100:NEXT J
120 NEXT I
130 LOCATE 10,21:PRINT "OTRA TIRADA? (S/N)"
140 GET A$:IF A$="" THEN GOTO 130
150 IF A$="S" THEN GOTO 50
160 IF A$<>"N" THEN GOTO 130
170 END
1000 REM * UNO *
1010 LOCATE 20,12:PRINT CHR$(113)
1020 RETURN
2000 REM * DOS *
2010 LOCATE 18,10:PRINT CHR$(113)
2020 LOCATE 22,14:PRINT CHR$(113)
2030 RETURN
3000 REM * TRES *
3010 LOCATE 18,10:PRINT CHR$(113)
3020 LOCATE 20,12:PRINT CHR$(113)
3030 LOCATE 22,14:PRINT CHR$(113)
3040 RETURN
4000 REM * CUATRO *
4010 LOCATE 18,10:PRINT CHR$(113);"   ";
     CHR$(113)
```

```
4020 LOCATE 18,14:PRINT CHR$(113);"   ";
     CHR$(113)
4030 RETURN
5000 REM * CINCO *
5010 LOCATE 18,10:PRINT CHR$(113);"   ";
     CHR$(113)
5020 LOCATE 20,12:PRINT CHR$(113)
5030 LOCATE 18,14:PRINT CHR$(113);"   ";
     CHR$(113)
5040 RETURN
6000 REM * SEIS *
6010 LOCATE 18,10:PRINT CHR$(113);"   ";
     CHR$(113)
6020 LOCATE 18,12:PRINT CHR$(113);"   ";
     CHR$(113)
6030 LOCATE 18,14:PRINT CHR$(113);"   ";
     CHR$(113)
6040 RETURN
```



Ver apéndice C.

MOVIMIENTOS ALEATORIOS

Vamos a conseguir ahora que el ordenador mueva una figura por la pantalla en la dirección que quiera, de modo que nosotros no podamos adivinar hacia dónde se va a desplazar en cada momento.

Para ello el ordenador va a generar dos números al azar: uno representará el desplazamiento horizontal y otro el vertical. Cada uno de estos dos números puede tomar tres valores distintos: 0, 1 ó 2. El 0 indica que no hay desplazamiento en esa dirección, el 1 indica un desplazamiento hacia la derecha (horizontal) o hacia abajo (vertical). Por último, el 2 señala un desplazamiento hacia la izquierda (horizontal) o hacia arriba (vertical).

```
10 REM **********************
20 REM *MOVIMIENTO ALEATORIO*
30 REM *      SPECTRUM      *
40 REM **********************
50 RANDOMIZE 0
60 LET X=11: LET Y=15
70 PRINT AT X-1,Y;"   ";AT X,Y;" # ";
   AT X+1,Y;"   "
80 LET H=INT (RND*3)
90 IF H<>0 THEN  GO SUB (H+1)*100
```

```
100 LET V=INT (RND*3)
110 IF V<>0 THEN  GO SUB (V+3)*100
120 PAUSE 10
130 GO TO 70
200 REM * DERECHA *
210 LET Y=Y+1
220 IF Y>29 THEN  LET Y=29
230 RETURN
300 REM * IZQUIERDA *
310 LET Y=Y-1
320 IF Y<0 THEN  LET Y=0
330 RETURN
400 REM * ABAJO *
410 LET X=X+1
420 IF X>20 THEN  LET X=20
430 RETURN
500 REM * ARRIBA *
510 LET X=X-1
520 IF X<1 THEN  LET X=1
530 RETURN
```

El programa 6.3 hace que un carácter se desplace de forma aleatoria por la pantalla, dando la impresión de que no sabe muy bien adónde quiere ir o de que está mareado. Cada uno de los desplazamientos posibles (derecha, izquierda, abajo o arriba) están estructurados en una subrutina independiente (200, 300, 400 y 500).

El programa 6.4 consigue igualmente el efecto anterior, pero en el IBM.

```
10 REM *************************
20 REM * MOVIMIENTO ALEATORIO *
30 REM *           IBM          *
40 REM *************************
50 SCREEN 0:WIDTH 40:CLS
60 RANDOMIZE TIMER
70 X=13:Y=20
80 LOCATE X-1,Y:PRINT "   "
90 LOCATE X,Y:PRINT " ";CHR$(15);" "
100 LOCATE X+1,Y:PRINT "   "
110 H=INT(RND*3)
120 IF H<>0 THEN ON H GOSUB 200,300
130 V=INT(RND*3)
140 IF V<>0 THEN ON V GOSUB 400,500
150 FOR I=1 TO 500:NEXT
160 GOTO 80
200 REM * DERECHA *
```

```
210 Y=Y+1
220 IF Y>38 THEN Y=38
230 RETURN
300 REM * IZQUIERDA *
310 Y=Y-1
320 IF Y<1 THEN Y=1
330 RETURN
400 REM *ABAJO *
410 X=X+1
420 IF X>21 THEN X=21
430 RETURN
500 REM * ARRIBA *
510 X=X-1
520 IF X<2 THEN X=2
530 RETURN
```

Para MSX, ver apéndice B.

Tanto en el programa 6.4 como en el 6.3 está previsto que si la figura en movimiento llega a alguno de los límites de la pantalla, choque contra este borde y no pueda sobrepasarlo.



EL DUENDE MISTERIOSO

Vamos a ver ahora cómo, mediante la función RND, podemos hacer que una figura aparezca y desaparezca en distintos puntos de la pantalla. Para ello vamos a desarrollar un programa de un juego en el que un hombrecito, controlado por nosotros, trata de alcanzar a un misterioso duende que tan pronto aparece en un sitio como en otro (el ordenador «decide» dónde va a aparecer). En primer lugar, tenemos que definir los caracteres del hombrecito y del duende.

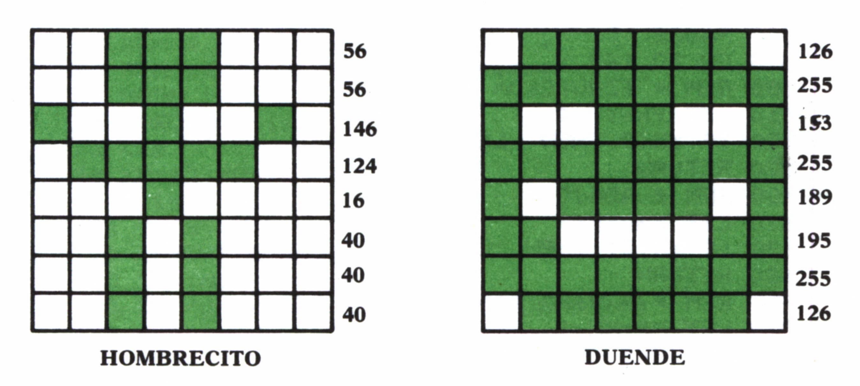


Fig. 6.2. Diseño de hombrecito y duende en la malla de 8 por 8.

La figura 6.2 muestra los esquemas del hombrecito y el duende en la malla de 8 por 8. Al lado están indicados los números decimales equivalentes a los binarios que hay que introducir en cada byte.

```
 10 REM ********************
 20 REM * DUENDE MISTERIOSO *
 30 REM *       SPECTRUM     *
 40 REM ********************
 50 DATA 56,56,146,124,16,40,40,40
 60 DATA 126,255,153,255,189,195,255,126
 70 FOR I=0 TO 15
 80 READ N
 90 POKE USR "A"+I,N
100 NEXT I
110 INK 1: BORDER 1: PAPER 6
120 CLS : RANDOMIZE 0
130 LET XH=11: LET YH=16
140 LET C=INT (RND*3)
150 LET T=INT (RND*10)+1
160 LET XD=INT (RND*22)
170 LET YD=INT (RND*32)
180 IF XD=XH AND YD=YH THEN  GO TO 160
190 PRINT  INK C;AT XD,YD;CHR$ 145
200 FOR I=1 TO T
210 PRINT AT XH,YH;CHR$ 144
220 PAUSE 10
230 IF XD=XH-1 AND YD=YH OR XD=XH+1 AND YD=YH
    OR XD=XH AND YD=YH-1 OR XD=XH AND YD=YH+1
    THEN  GO TO 1000
240 LET A$=INKEY$: IF A$="" THEN  GO TO 240
250 IF CODE A$<53 OR CODE A$>56 THEN  GO TO 240
260 PRINT AT XH,YH;" "
270 GO SUB VAL (A$)*100
280 NEXT I
290 PRINT AT XD,YD;" "
300 PAUSE 10
310 GO TO 140
500 REM * IZQUIERDA *
510 LET YH=YH-1
520 IF YH<0 THEN  LET YH=0
530 RETURN
600 REM * ABAJO *
610 LET XH=XH+1
620 IF XH>21 THEN  LET XH=21
630 RETURN
700 REM * ARRIBA *
710 LET XH=XH-1
720 IF XH<0 THEN  LET XH=0
```

```
730 RETURN
800 REM * DERECHA *
810 LET YH=YH+1
820 IF YH>31 THEN  LET YH=31
830 RETURN
```

Ya tenemos nuestro juego en el programa 6.5. Las líneas DATA almacenan los números necesarios para la definición de los personajes, mientras que el bucle FOR-NEXT de las líneas 70 a 100 introduce dichos números en las direcciones de memoria correspondientes a las teclas A (hombrecito) y B (duende). En la línea 110 se establecen los colores del hombrecito (tinta), el fondo y el borde. La línea 130 determina las coordenadas iniciales del hombrecito (centro de la pantalla). La línea 140 sirve para generar un número al azar entre 0 y 8 que determina el color que tendrá el duende en cada aparición. Por supuesto, si el color del duende coincide con el del fondo, éste se tornará invisible. En la línea siguiente se genera un número al azar que determina el tiempo de que dispone el hombrecito para aproximarse al duende antes de que desaparezca. Las líneas 160 y 170 sirven para generar al azar las dos coordenadas de posición del duende. El bucle FOR-NEXT de las líneas 200 a 280 nos permite mover al hombrecito tantos pasos como indica la variable T. Si en uno de estos desplazamientos consigue alcanzar al duende, finaliza el juego y se detiene el programa. Esto se comprueba con la condición de la línea 230. Los desplazamientos del hombrecito se controlan con las flechas del teclado y cada sentido está estructurado en una subrutina distinta. Si el hombrecito llega a un extremo de la pantalla, choca con el borde.

```
10 REM ************************
20 REM *  DUENDE MISTERIOSO   *
30 REM *       AMSTRAD        *
40 REM ************************
50 MODE 1
60 BORDER 0:INK 0,16
70 CLS
80 XH=12:YH=20
90 C=INT(RND*27)
100 T=INT(RND*15)+5
110 XD=INT(RND*22)+1
120 YD=INT(RND*40)+1
130 IF XD=XH AND YD=YH THEN GOTO 110
140 PEN 1:INK 1,C
150 LOCATE YD,XD:PRINT CHR$(224)
```

```
160 PEN 2:INK 2,1
170 FOR I=1 TO T
180 LOCATE YH,XH:PRINT CHR$(248)
190 FOR J=1 TO 100:NEXT
200 IF XD=XH-1 AND YD=YH OR XD=XH+1 AND YD=YH
    OR XD=XH AND YD=YH-1 OR XD=XH AND YD=YH+1
    THEN END
210 A$=INKEY$:IF A$="" THEN GOTO 210
220 IF ASC(A$)<240 OR ASC(A$)>243 THEN GOTO 210
230 LOCATE YH,XH:PRINT " "
240 ON ASC(A$)-239 GOSUB 500,600,700,800
250 NEXT
260 LOCATE YD,XD:PRINT " "
270 FOR J=1 TO 100:NEXT
280 GOTO 90
500 REM * ARRIBA *
510 XH=XH-1
520 IF XH<1 THEN XH=1
530 RETURN
600 REM * ABAJO *
610 XH=XH+1
620 IF XH>22 THEN XH=22
630 RETURN
700 REM * IZQUIERDA *
710 YH=YH-1
720 IF YH<1 THEN YH=1
730 RETURN
800 REM * DERECHA *
810 YH=YH+1
820 IF YH>40 THEN YH=40
830 RETURN
```

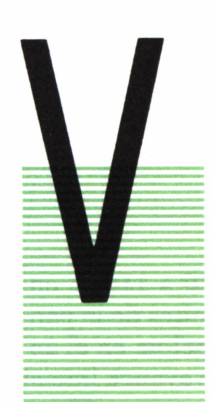
El programa 6.6 es la versión del Duende Misterioso, pero para el Amstrad. En este caso no hace falta definir los caracteres de los personajes, ya que se encuentran definidos en los códigos 248 (hombrecito) y 224 (duende).

Finalmente la figura 6.3 muestra una posible situación de la pantalla durante la ejecución del programa.



Fig. 6.3. El hombrecito trata de alcanzar al duende misterioso.

EFECTOS ESPECIALES 7



INTRODUCCION



VAMOS a dedicar el presente capítulo al estudio y desarrollo de los principales efectos especiales que podemos encontrar en cualquier videojuego, desde el más sencillo al más sofisticado. Todos hemos visto rebotar una pelota al chocar contra una pared o una raqueta, o el ataque del enemigo galáctico disparando proyectiles sin cesar o incluso distintos personajes moviéndose simultáneamente por la pantalla. Todos estos efectos, aparentemente complicados, pueden resultar muy sencillos de desarrollar con sólo comprender unas pocas ideas básicas.



CHOQUE Y REBOTE

Ya vimos en el capítulo 3 algunas ideas sobre el efecto de choque y rebote. Sin embargo, en aquella ocasión estudiamos el caso más simple: una pelota que se movía horizontalmente rebotando entre dos paredes. En este capítulo vamos a profundizar un poco más. En primer lugar vamos a desarrollar un programa que nos permite visualizar una pelota rebotando contra cuatro paredes, como si estuviera dentro de una caja.

Para ello primero hay que trazar las cuatro paredes que definen el recinto en el que se mueve la pelota. A continuación tenemos que establecer las velocidades horizontal y vertical de la pelota y las condiciones de rebote.

```
10 REM ******************
20 REM *PELOTA REBOTANDO*
30 REM *    SPECTRUM    *
40 REM ******************
```

```
 50 INK 0: BORDER 2: PAPER 5
 60 CLS
 70 FOR I=0 TO 31
 80 PRINT AT 0,I;CHR$ 143;AT 21,I;CHR$ 143
 90 IF I>21 THEN  GO TO 110
100 PRINT AT I,0;CHR$ 143;AT I,31;CHR$ 143
110 NEXT I
120 LET X=1: LET Y=1
130 LET V=1: LET H=1
140 PRINT AT X,Y;" "
150 LET X=X+V: LET Y=Y+H
160 IF X<2 OR X>19 THEN  LET V=-V
170 IF Y<2 OR Y>29 THEN  LET H=-H
180 PRINT AT X,Y;CHR$ 143
190 PAUSE 3
200 GO TO 140
```

En el programa 7.1 el bucle **FOR-NEXT** de las líneas 70 a 110 se encarga de dibujar las cuatro paredes coincidiendo con los límites de la pantalla. En la línea 120 establecemos la posición inicial de la pelota en el ángulo superior izquierdo. La línea 130 establece el sentido de avance inicial (hacia la derecha y hacia abajo), sin embargo, cada vez que se verifique una de las condiciones de las líneas 160 y 170, querrá decir que la pelota ha chocado contra una pared y por tanto tendrán que variar los sentidos de avance. Por ejemplo, si la pelota avanza hacia la dereha y hacia abajo y choca con la pared inferior, ahora hacia arriba, por el efecto del rebote. Esta idea se reproduce de forma esquemática en la figura 7.1.

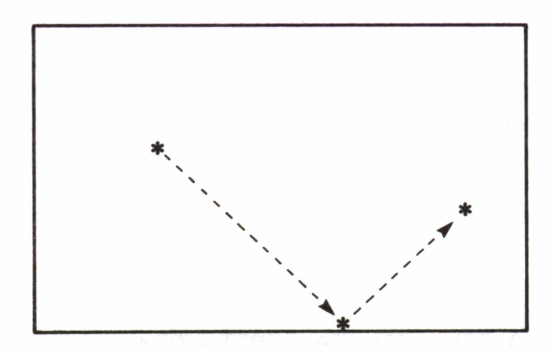


Fig. 7.1. Esquema de choque y rebote general.

El programa 7.2 simulta también el efecto de la pelota rebotando entre cuatro paredes, pero esta vez en el Commodore.

```
10 REM ***********************
20 REM *  PELOTA REBOTANDO   *
30 REM *     COMMODORE       *
40 REM ***********************
50 PRINT CHR$(147)
60 FOR I=0 TO 39
70 LOCATE I,0:PRINT CHR$(166)
80 LOCATE I,21:PRINT CHR$(166)
90 IF I>21 THEN GOTO 120
100 LOCATE 0,I:PRINT CHR$(166)
110 LOCATE 39,I:PRINT CHR$(166)
120 NEXT I
130 X=1:Y=1
140 V=1:H=1
150 LOCATE Y,X:" "
160 X=X+V:Y=Y+H
170 IF X<2 OR X>19 THEN V=-V
180 IF Y<2 OR Y>37 THEN H=-H
190 LOCATE Y,X:PRINT CHR$(113)
200 FOR I=1 TO 50:NEXT I
210 GOTO 150
```

Ver apéndice C.

Finalmente, la figura 7.2 muestra el estado de la pantalla en un momento de la ejecución de los programas 7.1 ó 7.2.

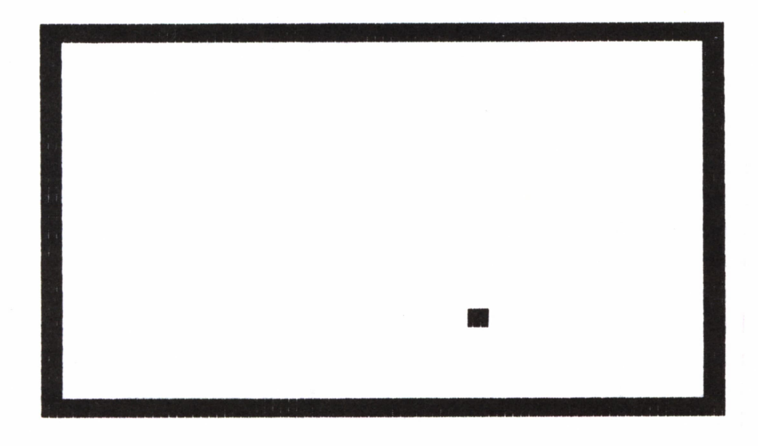


Fig. 7.2. La pelota rebota entre las cuatro paredes.

MOVIMIENTOS SIMULTANEOS

Podemos hacer que en la pantalla del ordenador aparezcan varias figuras moviéndose a la vez o, mejor dicho, que parezcan que se mueven simultáneamente.

En realidad el ordenador sólo puede ejecutar una orden en cada instante y para mover distintas figuras necesitaríamos varias órdenes. Lo que sucede es que estas órdenes las ejecuta tan de prisa que nosotros no podemos percibir que realmente los movimientos no son simultáneos.

Vamos a desarrollar, como ejemplo de lo dicho, un programa para simular carreras de caballos.

Los participantes en la carrera avanzarán en función de un número al azar, generado mediante RND. Dicho número determina el caballo que va a avanzar. El primer caballo que llegue al extremo derecho de la pantalla será el ganador.

Lo primero que tenemos que hacer es diseñar un caballo de carreras tal y como muestra la figura 7.3.

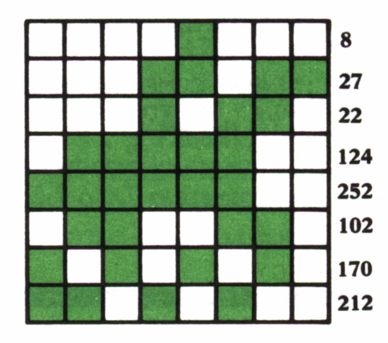


Fig. 7.3 Diseño de un caballo de carreras.

A la derecha de cada uno de los ocho bytes que componen el caballo (con jinete) figuran los ocho números decimales correspondientes a los binarios que tenemos que introducir en memoria para definir el caballo en la tecla de la A (código 144).

En el programa 7.3 hemos desarrollado la carrera para el Spectrum. De las líneas 70 a la 120 definimos el caballo. La línea 140 nos permite determinar el número de caballos que van a participar y que no podrán ser menos de 2 ni más de 9 (línea 150). En la línea 160 dimensionamos dos matrices. La primera para guardar los nombres de los caballos y la segunda para controlar el avance de cada uno. El bucle de las líneas 170 a 200 nos permite introducir los nombres de los caballos. El bucle FOR-NEXT de las

```
10 REM *********************
20 REM *CARRERA DE CABALLOS*
30 REM *     SPECTRUM      *
40 REM *********************
50 INK 0: BORDER 6: PAPER 4
60 CLS
70 REM *DEFINICION DEL CABALLO*
80 DATA 8,27,22,124,252,102,170,212
90 FOR I=0 TO 7
100 READ N
110 POKE USR "A"+I,N
120 NEXT I
130 REM *PREPARACION DE CABALLOS*
140 INPUT "NUMERO DE CABALLOS  ";C
150 IF C<2 OR C>9 THEN  GO TO 140
160 DIM N$(C,10): DIM A(C)
170 FOR I=1 TO C
180 INPUT "NOMBRE DEL CABALLO ";(I);N$(I)
190 LET A(I)=0
200 NEXT I
210 REM *COMIENZO DE LA CARRERA*
220 FOR I=1 TO C
230 PRINT I;CHR$ 144: PRINT
240 NEXT I
250 PRINT AT 20,1;"PARA COMENZAR,PULSE UNA TECLA"
260 IF INKEY$="" THEN  GO TO 260
270 FOR I=30 TO 20 STEP -1
280 BEEP 0.1,I
290 NEXT I
300 PRINT AT 20,1;"                             "
310 RANDOMIZE 0
320 LET J=INT (RND*C)+1
350 LET A(J)=A(J)+1
360 IF A(J)>30 THEN  GO TO 400
370 PRINT AT (J-1)*2,A(J);" ";CHR$ 144
390 GO TO 320
400 PAUSE 100
410 PRINT AT 19,6;"FINAL DE LA CARRERA"
420 PRINT AT 21,2;"CABALLO GANADOR: ";N$(J)
```

líneas 220 a 240 pone a todos los caballos en la línea de salida. En la línea 320 se genera un número al azar que determina el caballo que va a avanzar en cada momento. Este avance se hace efectivo en el bucle de las líneas 340 a 380. Además, en la línea 360 se controla cuando llega el primer caballo a la línea de meta.

El programa 7.4 simula la carrera de caballos en el Amstrad. Podemos

```
10 REM ************************
20 REM * CARRERA DE CABALLOS *
30 REM *        AMSTRAD        *
40 REM ************************
50 INK 0,9:INK 1,0:BORDER 22
60 MODE 1:CLS
70 REM * DEFINICION DEL CABALLO *
80 SYMBOL 240,8,27,22,124,252,102,170,212
90 REM * PREPARACION DE CABALLOS *
100 INPUT "NUMERO DE CABALLOS";C
110 CLS
120 IF C<2 OR C>9 THEN GOTO 100
130 DIM N$(C):DIM A(C)
140 FOR I=1 TO C
150 INPUT "NOMBRE DEL CABALLO ";N$(I)
160 LET A(I)=1
170 CLS
180 NEXT
190 REM * COMIENZO DE LA CARRERA *
200 FOR I=1 TO C
210 PRINT:PRINT I;CHR$(240)
220 NEXT
230 LOCATE 5,22:PRINT
    "PARA COMENZAR, PULSE UNA TECLA"
240 IF INKEY$="" THEN GOTO 240
250 LOCATE 5,22:PRINT SPC(30)
260 RANDOMIZE TIME
270 J=INT(RND*C)+1
300 A(J)=A(J)+1
310 IF A(J)>37 THEN GOTO 350
315 U=J*2:V=A(J)+2
320 LOCATE V,U:PRINT" ";CHR$(240)
340 GOTO 270
350 FOR I=1 TO 100:NEXT
360 LOCATE 10,20:PRINT"FINAL DE LA CARRERA"
370 LOCATE 6,22:PRINT"CABALLO GANADOR ";N$(J)
```

observar que cuanto mayor sea el número de participantes más se notará que los movimientos no son simultáneos realmente.

La figura 7.4 muestra el resultado de una posible ejecución, en plena carrera.



Fig. 7.4. La carrera de caballos está muy disputada.



DISPAROS

Vamos a estudiar ahora otro efecto interesante como son los disparos. Para ello vamos a desarrollar un programa en el que un vaquero formado por caracteres redefinidos tiene que acertar a una serie de blancos.

```
 10 REM ****************
 20 REM *   DISPAROS   *
 30 REM *   SPECTRUM   *
 40 REM ****************
 50 REM *DEFINICION DE VAQUERO*
 60 DATA 0,0,60,60,255,60,62,60
 70 DATA 24,60,126,127,127,126,126,126
 80 DATA 0,0,0,0,128,254,120,0
 90 DATA 126,126,102,102,102,102,102,119
100 FOR I=0 TO 31
110 READ N
120 POKE USR "A"+I,N
130 NEXT I
140 REM *SITUACION DE BLANCOS*
150  INK 0: BORDER 1: PAPER 6: CLS
160 FOR I=1 TO 5
170 LET C=INT (RND*5)+1
180 LET XT=INT (RND*20)+1
190 LET YT=INT (RND*16)+15
200 PRINT  INK C;AT XT,YT;CHR$ 143
210 NEXT I
220 LET X=11: LET Y=1: LET D=0
230 PRINT AT X-1,Y;CHR$ 144
```

```
240 PRINT AT X,Y;CHR$ 145;CHR$ 146
250 PRINT AT X+1,Y;CHR$ 147
260 LET A$=INKEY$
270 IF A$="" THEN  GO TO 230
280 IF CODE A$<>54 AND CODE A$<>55 AND
   CODE A$<>56 THEN  GO TO 230
290 IF CODE A$<>56 THEN  PRINT AT X-1,Y;
   "  ";AT X,Y;"  ";AT X+1,Y;"  "
300 GO SUB VAL (A$)*100
310 PAUSE 5
320 GO TO 230
600 REM * ABAJO*
610 LET X=X+1
620 IF X>20 THEN  LET X=20
630 RETURN
700 REM * ARRIBA *
710 LET X=X-1
720 IF X<1 THEN  LET X=1
730 RETURN
800 REM * DISPARO *
810 IF D=5 THEN  GO TO 1000
820 LET YB=Y+2
830 PRINT AT X,YB;" ."
840 LET YB=YB+1
850 IF YB>30 THEN  PRINT AT X,YB;" ":
    LET D=D+1: GO TO 870
860 GO TO 830
870 RETURN
```

El programa 7.5 define al vaquero entre las líneas 50 y 130. El bucle **FOR-NEXT** de las líneas 160 a 210 sitúa en 5 posiciones de la pantalla, determinadas al azar (líneas 180 y 190), 5 cuadritos de colores, también al azar (línea 170).

La función **INKEY$** de la línea 260 nos permite disparar apretando el 8 (código 56) o mover al vaquero de arriba a abajo y viceversa con las teclas 5 y 6 (líneas 280 y 300). Finalmente, cada uno de los movimientos (arriba, abajo y disparo) está estructurado en una subrutina independiente.

La figura 7.6 muestra el momento en el que el vaquero acaba de disparar.

Por último, el programa 7.6 produce un efecto similar en el **IBM**, sólo que ahora la figura que dispara es un tanque.

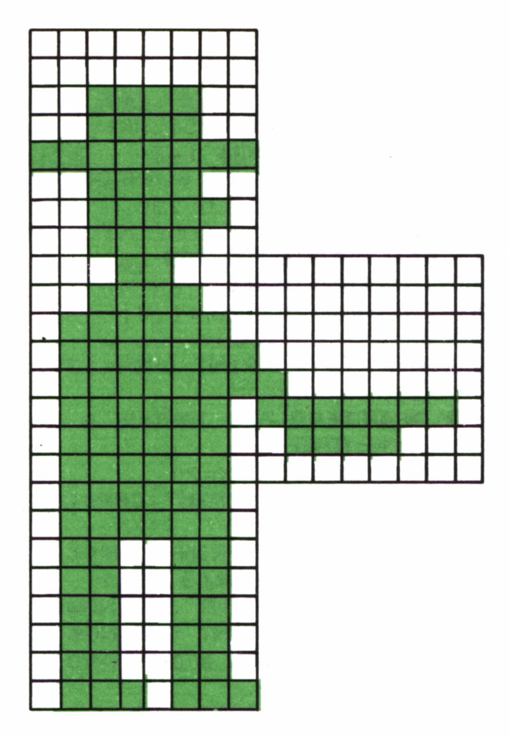


Fig. 7.5. Diseño de vaquero utilizando cuatro caracteres.



Fig. 7.6. El vaquero sólo puede hacer cinco disparos para acertar los cinco blancos.

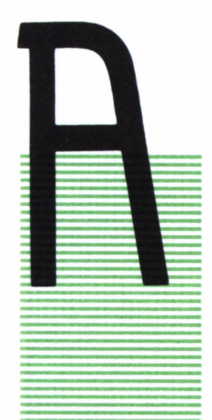
```
10 REM ****************
20 REM *   DISPAROS    *
30 REM *     IBM       *
40 REM ****************
```

```
50 SCREEN 0:WIDTH 40:KEY OFF
60 COLOR 4,2,10:CLS
70 RANDOMIZE TIMER
80 REM * SITUACION DE BLANCOS *
90 FOR I=1 TO 5
100 C=INT(RND*16)
110 IF C=2 THEN GOTO 100
120 XT=INT(RND*18)+2
130 YT=INT(RND*20)+20
140 COLOR C
150 LOCATE XT,YT:PRINT CHR$(177)
160 NEXT
170 REM * DISEÑO DEL TANQUE *
180 X=11:Y=2:D=0
190 COLOR 4
200 LOCATE X-2,Y+4:PRINT CHR$(218);
    CHR$(196)
210 LOCATE X-1,Y+4:PRINT CHR$(219);
    CHR$(219);STRING$(3,205)
220 LOCATE X,Y+1:PRINT STRING$(8,219)
230 LOCATE X+1,Y:PRINT STRING$(10,219)
240 REM * MOVIMIENTO Y DISPARO *
250 A$=INKEY$
260 IF A$="" THEN GOTO 250
270 IF ASC(A$)<>49 AND ASC(A$)<>50
    AND ASC(A$)<>48 THEN GOTO 250
280 IF ASC(A$)=48 THEN GOTO 330
290 LOCATE X-2,Y+4:PRINT "  "
300 LOCATE X-1,Y+4:PRINT SPC(5)
310 LOCATE X,Y+1:PRINT SPC(8)
320 LOCATE X+1,Y:PRINT SPC(10)
330 ON VAL(A$)+1 GOSUB 600,700,800
340 FOR I=1 TO 50:NEXT
350 GOTO 200
600 REM * DISPARO *
610 IF D=5 THEN END
620 YB=Y+9
630 LOCATE X-1,YB:PRINT " ";CHR$(249)
640 YB=YB+1
650 IF YB>38 THEN LOCATE X-1,YB:
    PRINT "  ":GOTO 680
660 FOR I=1 TO 100:NEXT
670 GOTO 630
680 D=D+1
690 RETURN
700 REM * ABAJO *
710 X=X+1
720 IF X>20 THEN X=20
```

```
730 RETURN
800 REM * ARRIBA *
810 X=X-1
820 IF X<3 THEN X=3
830 RETURN
```

Programa 7.6. Para MSX, ver apéndice B.

MOVIMIENTO EN ALTA RESOLUCION 8



INTRODUCCION



A lo largo de todo el libro hemos aprendido los secretos de todo tipo de movimientos sobre la pantalla de baja resolución, que es sobre la que se diseñan la gran mayoría de los videojuegos. El motivo de esta elección es bien sencillo: el ordenador tiene que trabajar mucho más en alta resolución y, por tanto, no se puede obtener la rapidez que requiere cualquier videojuego. Los movimientos en alta resolución resultan lentos y, por consiguiente, menos atractivos que en baja resolución. Sin embargo, vamos a dedicar el presente capítulo a dar unas nociones básicas sobre los movimientos en pantalla gráfica, aunque realmente vamos a poder comprobar que los fundamentos son los mismos que hemos visto hasta ahora.



DESCRIPCION DE LA PANTALLA

La pantalla de alta resolución está formada por una retícula de pequeños puntos denominados pixels que no son más que los puntos que constituyen la malla de 8 por 8 de cada posición de la pantalla de texto. Por tanto para saber las dimensiones de la pantalla gráfica de un ordenador no tenemos más que multiplicar por 8 el número de filas y el número de columnas que constituyen la pantalla de baja resolución. De este modo podemos comprobar que el Spectrum dispone de una pantalla gráfica de 176 filas por 256 columnas. El Amstrad dispone de tres pantallas distintas: 200 filas por 160 columnas, 200 filas por 320 columnas y 200 filas por 640 columnas, que se corresponden con los modos 0, 1 y 2 respectivamente. El Commodore tiene 200 filas por 320 columnas y finalmente, el IBM cuenta con dos pantallas de 200 filas por 320 columnas y 200 filas por 640 colum-

nas que se obtienen respectivamente con las sentencias SCREEN 1 y SCREEN 2.

Normalmente el origen de coordenadas se sitúa en el ángulo inferior izquierdo en el punto 0,0, excepto el IBM que tiene el origen en el ángulo superior izquierdo.

175

80

100,80

0,0

100

255

Fig. 8.1. Pantalla de alta resolución del Spectrum.

La figura 8.1 muestra la pantalla de alta resolución del Spectrum con el punto de coordenadas 100, 80 situado en la misma.



MOVIMIENTO DE UN PUNTO

En primer lugar debemos advertir que las sentencias gráficas suelen variar de una marca de ordenadores a otra. Evidentemente en un sólo capítulo no podemos describir las sentencias de cada uno de los cuatro ordenadores en estudio por tanto vamos a centrarnos en el Spectrum. En cualquier caso los programas que se presentan son sencillos de trasladar a cualquier ordenador sin más que sustituir las sentencias gráficas por las que

aparezcan en el Manual de instrucciones correspondiente, además de considerar las dimensiones de la pantalla sobre la que estemos trabajando.

Vamos a comenzar por el movimiento de un punto sobre una recta horizontal de izquierda a derecha. Este movimiento lo va a controlar el ordenador. Los fundamentos son los mismos que los vistos en el capítulo 3 para el movimiento de un carácter, sólo que trasladados a alta resolución.

```
 10 REM ********************
 20 REM *PUNTO EN MOVIMIENTO*
 30 REM *      SPECTRUM     *
 40 REM ********************
 50 LET X=0: LET Y=88
 60 PLOT X,Y
 80 PLOT  OVER 1;X,Y
 90 LET X=X+1
100 IF X>255 THEN  LET X=0
110 GO TO 60
```

En el programa 8.1 la sentencia PLOT de la línea 60 se encarga de imprimir un punto en la posición (X, Y) de la pantalla. En cambio, la línea 80 borra dicho punto. Esto es debido a la sentencia OVER1, que hace que si el punto de la posición (9X,Y) está encendido (color de la tinta) se apague (color del papel) y viceversa. La función OVER 1 se desactiva con OVER 0. El resto del programa no necesita comentario, puesto que es una aplicación de lo visto en el capítulo 3.



CONTROL DEL MOVIMIENTO

Si queremos ser nosotros los que controlemos el movimiento del punto no tenemos más que introducir la función INKEY$ tal y como vimos en el capítulo 4. Vamos a desarrollar aquí el programa más completo que nos permitirá mover un punto en los cuatro sentidos principales utilizando las cuatro flechas de movimiento del cursor. Cuando el punto llegue a un extremo de la pantalla chocará con éste.

```
10 REM *******************
20 REM *CONTROL DEL PUNTO*
30 REM *     SPECTRUM    *
40 REM *******************
```

```
 50 INK 0: BORDER 1: PAPER 6
 60 CLS
 70 LET X=128: LET Y=88
 80 PLOT X,Y
100 LET A$=INKEY$: IF A$="" THEN  GO TO 100
110 IF CODE A$<53 OR CODE A$>56 THEN  GO TO 100
120 PLOT  OVER 1;X,Y
130 GO SUB VAL (A$)*100
140 GO TO 80
500 REM * IZQUIERDA *
510 LET X=X-1
520 IF X<0 THEN  LET X=0
530 RETURN
600 REM * ABAJO *
610 LET Y=Y-1
620 IF Y<0 THEN  LET Y=0
630 RETURN
700 REM * ARRIBA *
710 LET Y=Y+1
720 IF Y>175 THEN  LET Y=175
730 RETURN
800 REM * DERECHA *
810 LET X=X+1
820 IF X>255 THEN  LET X=255
830 RETURN
```

El programa 8.2 no es más que una traducción a alta resolución del programa 4.3.



CHOQUE Y REBOTE

Finalmente vamos a ver un ejemplo de un punto rebotando dentro de una circunferencia. Para averiguar cuándo el punto llega a tocar la circunferencia utilizamos la función POINT, que tiene el siguiente formato:

POINT (X,Y)

donde *X* e *Y* son las coordenadas de un punto de la pantalla. POINT devolverá un 1 si dicho punto está encendido (color de la tinta) y un 0 si el punto está apagado (color del papel).

```
 10 REM ******************
 20 REM * CHOQUE Y REBOTE *
 30 REM *      SPECTRUM    *
 40 REM ******************
 50 INK 6: PAPER 1: BORDER 5
 60 CLS
 70 CIRCLE 128,88,70
 80 LET X=128: LET Y=88: LET A=1
 90 PLOT X,Y
100 PLOT  OVER 1;X,Y
110 IF POINT (X+A,Y)=1 THEN  LET A=-A
120 LET X=X+A
130 GO TO 90
```

La figura 8.2 muestra un aspecto de la pantalla durante la ejecución del programa 8.3.

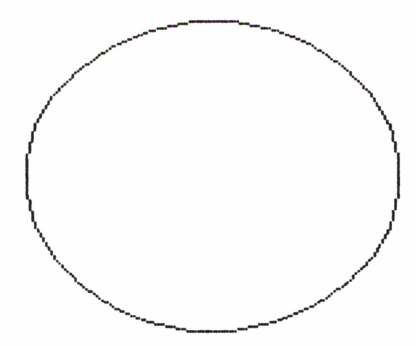


Fig. 8.2. El punto oscila de izquierda a derecha y viceversa dentro de la cincunferencia.

APENDICES

APENDICE A

PRINCIPALES VARIACIONES ENTRE LOS ORDENADORES SINCLAIR, AMSTRAD, COMMODORE E IBM

	SINCLAIR-SPECTRUM	AMSTRAD	COMMODORE	IBM
NUMERO DE FILAS	22	25	25	25
NUMERO DE COLUMNAS	32	20 40 80	40	40 80
NUMERACION DE FILAS	0...21	1...25	0...24	1...25
NUMERACION DE COLUMNAS	0...31	1...20 1...40 1...80	0...39	1...40 1...80
NUMERO DE ZONAS DE PANTALLA	2	1 3 6	4	2 5
NUMERO DE COLUMNA DE INICIO DE ZONAS	0, 16	1 1,14,27 1,14,27, 40,53,66	0,10,20 30	1,15 1,15, 29,43, 57
CAMBIO DE MODO DE PANTALLA	______	MODE 0 MODE 1 MODE 2	______	SCREEN 0: WIDTH 40 SCREEN 0: WIDTH 80
BORRADO DE PANTALLA	CLS	CLS	PRINT CHR$ (147)	CLS
SITUACION EN PANTALLA	AT F,C	LOCATE C,F	LOCATE C,F	LOCATE F,C
SUBRUTINAS	GOSUB "expresion"	ON- GOSUB	ON- GOSUB	ON- GOSUB



TABLA DE CONVERSION DECIMAL-BINARIO

0 =	00000000
1 =	00000001
2 =	00000010
3 =	00000011
4 =	00000100
5 =	00000101
6 =	00000110
7 =	00000111
8 =	00001000
9 =	00001001
10 =	00001010
11 =	00001011
12 =	00001100
13 =	00001101
14 =	00001110
15 =	00001111
16 =	00010000
17 =	00010001
18 =	00010010
19 =	00010011
20 =	00010100
21 =	00010101
22 =	00010110
23 =	00010111
24 =	00011000
25 =	00011001
26 =	00011010
27 =	00011011
28 =	00011100
29 =	00011101
30 =	00011110
31 =	00011111
32 =	00100000
33 =	00100001
34 =	00100010
35 =	00100011
36 =	00100100
37 =	00100101
38 =	00100110
39 =	00100111
40 =	00101000
41 =	00101001
42 =	00101010
43 =	00101011
44 =	00101100
45 =	00101101
46 =	00101110
47 =	00101111
48 =	00110000
49 =	00110001
50 =	00110010
51 =	00110011
52 =	00110100
53 =	00110101
54 =	00110110
55 =	00110111
56 =	00111000
57 =	00111001
58 =	00111010
59 =	00111011
60 =	00111100
61 =	00111101
62 =	00111110
63 =	00111111
64 =	01000000
65 =	01000001
66 =	01000010
67 =	01000011
68 =	01000100
69 =	01000101
70 =	01000110
71 =	01000111
72 =	01001000
73 =	01001001
74 =	01001010
75 =	01001011
76 =	01001100
77 =	01001101
78 =	01001110
79 =	01001111
80 =	01010000
81 =	01010001
82 =	01010010
83 =	01010011
84 =	01010100
85 =	01010101
86 =	01010110
87 =	01010111
88 =	01011000
89 =	01011001
90 =	01011010
91 =	01011011
92 =	01011100
93 =	01011101
94 =	01011110
95 =	01011111
96 =	01100000
97 =	01100001
98 =	01100010
99 =	01100011
100 =	01100100
101 =	01100101
102 =	01100110
103 =	01100111
104 =	01101000
105 =	01101001
106 =	01101010
107 =	01101011
108 =	01101100
109 =	01101101
110 =	01101110
111 =	01101111
112 =	01110000
113 =	01110001
114 =	01110010
115 =	01110011
116 =	01110100
117 =	01110101
118 =	01110110
119 =	01110111
120 =	01111000
121 =	01111001
122 =	01111010
123 =	01111011
124 =	01111100
125 =	01111101
126 =	01111110
127 =	01111111
128 =	10000000
129 =	10000001
130 =	10000010
131 =	10000011
132 =	10000100
133 =	10000101
134 =	10000110
135 =	10000111
136 =	10001000
137 =	10001001

138 = 10001010
139 = 10001011
140 = 10001100
141 = 10001101
142 = 10001110
143 = 10001111
144 = 10010000
145 = 10010001
146 = 10010010
147 = 10010011
148 = 10010100
149 = 10010101
150 = 10010110
151 = 10010111
152 = 10011000
153 = 10011001
154 = 10011010
155 = 10011011
156 = 10011100
157 = 10011101
158 = 10011110
159 = 10011111
160 = 10100000
161 = 10100001
162 = 10100010
163 = 10100011
164 = 10100100
165 = 10100101
166 = 10100110
167 = 10100111
168 = 10101000
169 = 10101001
170 = 10101010
171 = 10101011
172 = 10101100
173 = 10101101
174 = 10101110
175 = 10101111
176 = 10110000
177 = 10110001
178 = 10110010
179 = 10110011
180 = 10110100
181 = 10110101
182 = 10110110
183 = 10110111
184 = 10111000
185 = 10111001
186 = 10111010
187 = 10111011
188 = 10111100
189 = 10111101
190 = 10111110
191 = 10111111
192 = 11000000
193 = 11000001
194 = 11000010
195 = 11000011
196 = 11000100
197 = 11000101
198 = 11000110
199 = 11000111
200 = 11001000
201 = 11001001
202 = 11001010
203 = 11001011
204 = 11001100
205 = 11001101
206 = 11001110
207 = 11001111
208 = 11010000
209 = 11010001
210 = 11010010
211 = 11010011
212 = 11010100
213 = 11010101
214 = 11010110
215 = 11010111
216 = 11011000
217 = 11011001
218 = 11011010
219 = 11011011
220 = 11011100
221 = 11011101
222 = 11011110
223 = 11011111
224 = 11100000
225 = 11100001
226 = 11100010
227 = 11100011
228 = 11100100
229 = 11100101
230 = 11100110
231 = 11100111
232 = 11101000
233 = 11101001
234 = 11101010
235 = 11101011
236 = 11101100
237 = 11101101
238 = 11101110
239 = 11101111
240 = 11110000
241 = 11110001
242 = 11110010
243 = 11110011
244 = 11110100
245 = 11110101
246 = 11110110
247 = 11110111
248 = 11111000
249 = 11111001
250 = 11111010
251 = 11111011
252 = 11111100
253 = 11111101
254 = 11111110
255 = 11111111

CODIGO ASCII DEL SPECTRUM

32	–		76	–	L	120	–	x
33	–	!	77	–	M	121	–	y
34	–	"	78	–	N	122	–	z
35	–	#	79	–	O	123	–	{
36	–	$	80	–	P	124	–		
37	–	%	81	–	Q	125	–	}
38	–	&	82	–	R	126	–	~
39	–	'	83	–	S	127	–	©
40	–	(	84	–	T	128	–	
41	–	)	85	–	U	129	–	▝
42	–	*	86	–	V	130	–	▘
43	–	+	87	–	W	131	–	▀
44	–	,	88	–	X	132	–	▗
45	–	-	89	–	Y	133	–	▐
46	–	.	90	–	Z	134	–	▚
47	–	/	91	–	[	135	–	▜
48	–	0	92	–	\	136	–	▖
49	–	1	93	–	]	137	–	▞
50	–	2	94	–	↑	138	–	▌
51	–	3	95	–	_	139	–	▛
52	–	4	96	–	£	140	–	▄
53	–	5	97	–	a	141	–	▟
54	–	6	98	–	b	142	–	▙
55	–	7	99	–	c	143	–	█
56	–	8	100	–	d	144	–	A
57	–	9	101	–	e	145	–	B
58	–	:	102	–	f	146	–	C
59	–	;	103	–	g	147	–	D
60	–	<	104	–	h	148	–	E
61	–	=	105	–	i	149	–	F
62	–	>	106	–	j	150	–	G
63	–	?	107	–	k	151	–	H
64	–	@	108	–	l	152	–	I
65	–	A	109	–	m	153	–	J
66	–	B	110	–	n	154	–	K
67	–	C	111	–	o	155	–	L
68	–	D	112	–	p	156	–	M
69	–	E	113	–	q	157	–	N
70	–	F	114	–	r	158	–	O
71	–	G	115	–	s	159	–	P
72	–	H	116	–	t	160	–	Q
73	–	I	117	–	u	161	–	R
74	–	J	118	–	v	162	–	S
75	–	K	119	–	w	163	–	T

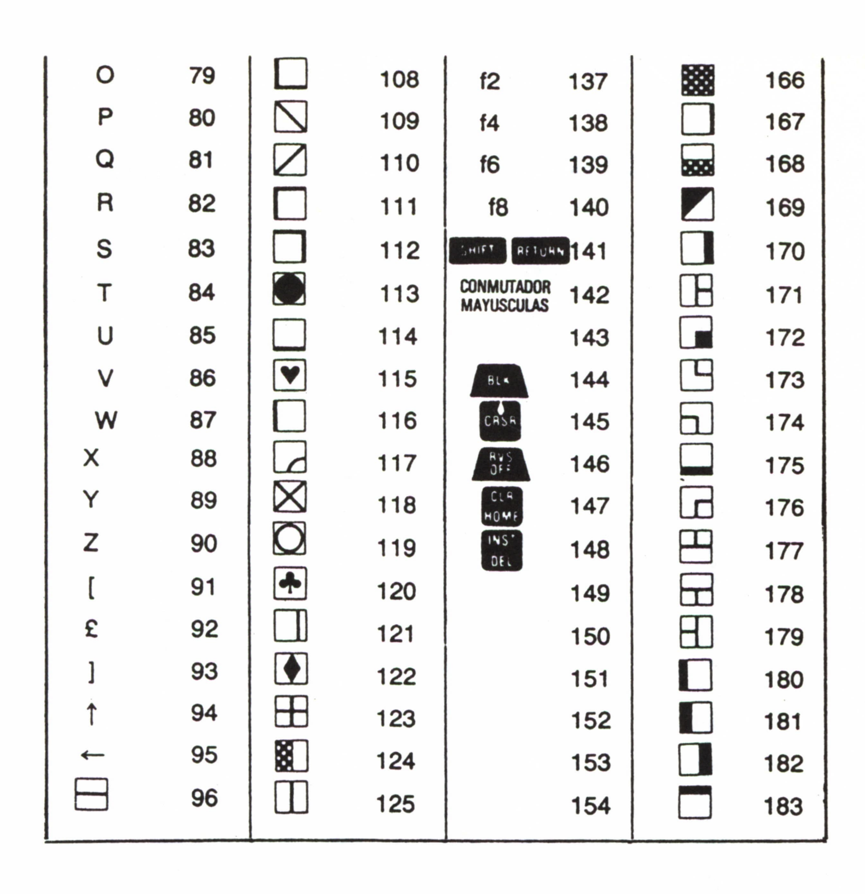
```
164  -  U
165  -  RND
166  -  INKEY$
167  -  PI
168  -  FN
169  -  POINT
170  -  SCREEN$
171  -  ATTR
172  -  AT
173  -  TAB
174  -  VAL$
175  -  CODE
176  -  VAL
177  -  LEN
178  -  SIN
179  -  COS
180  -  TAN
181  -  ASN
182  -  ACS
183  -  ATN
184  -  LN
185  -  EXP
186  -  INT
187  -  SQR
188  -  SGN
189  -  ABS
190  -  PEEK
191  -  IN
192  -  USR
193  -  STR$
194  -  CHR$
195  -  NOT
196  -  BIN
197  -  OR
198  -  AND
199  -  <=
200  -  >=
201  -  <>
202  -  LINE
203  -  THEN
204  -  TO
205  -  STEP
206  -  DEF FN
207  -  CAT
208  -  FORMAT
209  -  MOVE
210  -  ERASE
211  -  OPEN #
212  -  CLOSE #
213  -  MERGE
214  -  VERIFY
215  -  BEEP
216  -  CIRCLE
217  -  INK
218  -  PAPER
219  -  FLASH
220  -  BRIGHT
221  -  INVERSE
222  -  OVER
223  -  OUT
224  -  LPRINT
225  -  LLIST
226  -  STOP
227  -  READ
228  -  DATA
229  -  RESTORE
230  -  NEW
231  -  BORDER
232  -  CONTINUE
233  -  DIM
234  -  REM
235  -  FOR
236  -  GO TO
237  -  GO SUB
238  -  INPUT
239  -  LOAD
240  -  LIST
241  -  LET
242  -  PAUSE
243  -  NEXT
244  -  POKE
245  -  PRINT
246  -  PLOT
247  -  RUN
248  -  SAVE
249  -  RANDOMIZE
250  -  IF
251  -  CLS
252  -  DRAW
253  -  CLEAR
254  -  RETURN
255  -  COPY
```



CODIGO ASCII DEL COMMODORE

	0	CRSR ↓	17	"	34	3	51
	1	RVS ON	18	#	35	4	52
	2	CLR HOME	19	$	36	5	53
	3	INST DEL	20	%	37	6	54
	4		21	&	38	7	55
WHT	5		22	'	39	8	56
	6		23	(	40	9	57
	7		24	)	41	:	58
ANULAR SHIFT C=	8		25	*	42	;	59
PONER SHIFT C=	9		26	+	43	<	60
	10		27	,	44	=	61
	11	RED	28	–	45	>	62
	12	CRSR →	29	.	46	?	63
RETURN	13	GRN	30	/	47	@	64
CONMUTADOR MINUSCULAS	14	BLU	31	0	48	A	65
	15	SPACE	32	1	49	B	66
	16	!	33	2	50	C	67

D	68	♠	97	π	126		155
E	69		98		127	PUR	156
F	70		99		128	CRSR ←	157
G	71		100		129	YEL	158
H	72		101		130	CYN	159
I	73		102		131	SPACE	160
J	74		103		132		161
K	75		104	f1	133		162
L	76		105	f3	134		163
M	77		106	f5	135		164
N	78		107	f7	136		165



O	79		108	f2	137		166
P	80		109	f4	138		167
Q	81		110	f6	139		168
R	82		111	f8	140		169
S	83		112	SHIFT RETURN	141		170
T	84		113	CONMUTADOR MAYUSCULAS	142		171
U	85		114		143		172
V	86		115	BLK	144		173
W	87		116	CRSR	145		174
X	88		117	RVS OFF	146		175
Y	89		118	CLR HOME	147		176
Z	90		119	INST DEL	148		177
[	91		120		149		178
£	92		121		150		179
]	93		122		151		180
↑	94		123		152		181
←	95		124		153		182
	96		125		154		183

	184		186		188		190
	185		187		189		191

LOS CODIGOS	192 A 223	SON LOS MISMOS QUE	96 A 127
LOS CODIGOS	224 A 254	SON LOS MISMOS QUE	160 A 190
EL CODIGO	255	ES EL MISMO QUE	126



CODIGO ASCII DEL AMSTRAD

CARACTER	CHR$	CARACTER	CHR$	CARACTER	CHR$	CARACTER	CHR$
□	0		32	@	64	`	96
┌	1	!	33	A	65	a	97
⊥	2	"	34	B	66	b	98
┘	3	#	35	C	67	c	99
⚡	4	$	36	D	68	d	100
⊠	5	%	37	E	69	e	101
✓	6	&	38	F	70	f	102
Ω	7	'	39	G	71	g	103
←	8	(	40	H	72	h	104
→	9	)	41	I	73	i	105
↓	10	*	42	J	74	j	106
↑	11	+	43	K	75	k	107
↡	12	,	44	L	76	l	108
↵	13	-	45	M	77	m	109
⊗	14	.	46	N	78	n	110
⊙	15	/	47	O	79	o	111
⊟	16	0	48	P	80	p	112
◷	17	1	49	Q	81	q	113
◶	18	2	50	R	82	r	114
◵	19	3	51	S	83	s	115
◴	20	4	52	T	84	t	116
⌘	21	5	53	U	85	u	117
Π	22	6	54	V	86	v	118
┤	23	7	55	W	87	w	119
⌛	24	8	56	X	88	x	120
↕	25	9	57	Y	89	y	121
¿	26	:	58	Z	90	z	122
Θ	27	;	59	[	91	{	123
▣	28	<	60	\	92	|	124
▣	29	=	61	]	93	}	125
▣	30	>	62	↑	94	~	126
▣	31	?	63	_	95	▒	127

CARACTER	CHR$	CARACTER	CHR$	CARACTER	CHR$	CARACTER	CHR$
	128		160		192		224
	129		161		193		225
	130		162		194	♣	226
	131	£	163		195	♦	227
	132	©	164		196	♥	228
	133	¶	165		197	♠	229
	134	§	166		198	○	230
	135		167		199	●	231
	136	¼	168		200	□	232
	137	½	169		201	■	233
	138	¾	170		202	♂	234
	139	±	171		203	♀	235
	140	÷	172		204	♩	236
	141	¬	173		205	♪	237
	142	¿	174		206	☼	238
	143	¡	175		207		239
	144	α	176		208	↑	240
	145	β	177		209	↓	241
	146	γ	178		210	←	242
	147	δ	179		211	→	243
	148	ε	180		212	▲	244
	149	θ	181		213	▼	245
	150	λ	182		214	▶	246
	151	μ	183		215	◀	247
	152	π	184		216		248
	153	σ	185		217		249
	154	φ	186		218		250
	155	ψ	187		219		251
	156	χ	188		220		252
	157	ω	189		221		253
	158	Σ	190		222		254
	159	Ω	191		223		255



CODIGO ASCII DEL IBM

Valor ASCII	Carácter	Carácter de Control	Valor ASCII	Carácter
000	(nulo)	NUL	032	(espacio)
001	☺	SOH	033	!
002	●	STX	034	"
003	♥	ETX	035	#
004	♦	EOT	036	$
005	♣	ENQ	037	%
006	♠	ACK	038	&
007	(bip)	BEL	039	'
008	◘	BS	040	(
009	(tabulador)	HT	041	)
010	(avance de línea)	LF	042	*
011	(hogar)	VT	043	+
012	(avance de formulario)	FF	044	,
013	(retorno de carro)	CR	045	-
014	♫	SO	046	.
015	☼	SI	047	/
016	►	DLE	048	0
017	◄	DC1	049	1
018	↕	DC2	050	2
019	!!	DC3	051	3
020	¶	DC4	052	4
021	§	NAK	053	5
022	▬	SYN	054	6
023	↨	ETB	055	7
024	↑	CAN	056	8
025	↓	EM	057	9
026	→	SUB	058	:
027	←	ESC	059	;
028	(cursor a la derecha)	FS	060	<
029	(cursor a la izquierda)	GS	061	=
030	(cursor arriba)	RS	062	>
031	(cursor abajo)	US	063	?

Valor ASCII	Carácter	Valor ASCII	Carácter
064	@	096	‘
065	A	097	a
066	B	098	b
067	C	099	c
068	D	100	d
069	E	101	e
070	F	102	f
071	G	103	g
072	H	104	h
073	I	105	i
074	J	106	j
075	K	107	k
076	L	108	l
077	M	109	m
078	N	110	n
079	O	111	o
080	P	112	p
081	Q	113	q
082	R	114	r
083	S	115	s
084	T	116	t
085	U	117	u
086	V	118	v
087	W	119	w
088	X	120	x
089	Y	121	y
090	Z	122	z
091	[	123	{
092	\	124	¦
093	]	125	}
094	∧	126	~
095	_	127	⌂

Valor ASCII	Carácter	Valor ASCII	Carácter
128	Ç	160	á
129	ü	161	í
130	é	162	ó
131	â	163	ú
132	ä	164	ñ
133	à	165	Ñ
134	å	166	ª
135	ç	167	º
136	ê	168	¿
137	ë	169	⌐
138	è	170	¬
139	ï	171	½
140	î	172	¼
141	ì	173	¡
142	Ä	174	«
143	Å	175	»
144	É	176	░
145	æ	177	▒
146	Æ	178	▓
147	ô	179	│
148	ö	180	┤
149	ò	181	╡
150	û	182	╢
151	ù	183	╖
152	ÿ	184	╕
153	Ö	185	╣
154	Ü	186	║
155	¢	187	╗
156	£	188	╝
157	¥	189	╜
158	Pt	190	╛
159	ƒ	191	┐

Valor ASCII	Carácter	Valor ASCII	Carácter
192	└	224	α
193	┴	225	β
194	┬	226	Γ
195	├	227	π
196	─	228	Σ
197	┼	229	σ
198	╞	230	µ
199	╟	231	τ
200	╚	232	Φ
201	╔	233	Θ
202	╩	234	Ω
203	╦	235	δ
204	╠	236	∞
205	═	237	Ø
206	╬	238	ε
207	╧	239	∩
208	╨	240	≡
209	╤	241	±
210	╥	242	≥
211	╙	243	≤
212	╘	244	⌠
213	╒	245	⌡
214	╓	246	÷
215	╫	247	≈
216	╪	248	°
217	┘	249	•
218	┌	250	·
219	█	251	√
220	▄	252	^{n}
221	▌	253	^{2}
222	▐	254	■
223	▀	255	(espacio en blanco 'FF')

APENDICE B

En este apéndice se contemplan las modificaciones que hay que hacer en ciertos programas para que funcionen bajo el sistema M.S.X.

PROGRAMA 1.1: Para M.S.X. e I.B.M. cambiar las línea 60 por 'SCREEN N'

PROGRAMA 2.3: Para el sistema M.S.X. el programa quedaría así.

```
10 REM **********************
20 REM * CABALLO CON JINETE *
30 REM **********************
40 REM VERSION M.S.X
50 SCREEN 2:CLS
60 SPRITE$(1)=CHR$(240)+CHR$(16)+CHR$(16)+CHR$(19)+CHR$(125)+CHR$(252)
  +CHR$(84)+CHR$(68)+CHR$(68)
70 PUTSPRITE1,(100,100),,1
80 GOTO 80
```

PROGRAMA 3.2: Para M.S.X cambiar la línea 60 por 'SCREEN 0'

PROGRAMA 3.8: Para M.S.X. e I.B.M. cambiar la línea 50 por 'SCREEN 0:CLS'

PROGRAMA 4.4: Para M.S.X. e I.B.M. cambiar la línea 50 por 'SCREEN 0:CLS'

PROGRAMA 5.1: Para M.S.X. cambiar la línea 200 por:

```
200 FOR I=1 TO 14:A$="":FOR J=1 TO 8
```

y las líneas 220 y 230

```
220 A$=A$+CHR$(N)
230 NEXT J: SPRITE$(I)=A$:NEXT I
```

PROGRAMA 5.5: Para M.S.X. variar las siguientes líneas:

```
340 X=100:Y=100
360 PUTSPRITE 1,(X,Y),,1
370 PUTSPRITE 2,(X,Y+8),,5
380 PUTSPRITE 3,(X,Y+16),,10
400 PUTSPRITE 1,(X,Y),,2
410 PUTSPRITE 2,(X,Y+8),,4
430 PUTSPRITE 1,(X,Y),,1
440 PUTSPRITE 2,(X,Y+8),,5
460 PUTSPRITE 1,(X,Y),,3
470 PUTSPRITE 2,(X,Y+8),,4
610 PUTSPRITE 1,(X,Y),,0
620 PUTSPRITE 2,(X,Y+8),,0
630 PUTSPRITE 3,(X,Y+16),,0
660 PUTSPRITE 1,(X,Y),,2
670 PUTSPRITE 2,(X,Y+8),,6
680 PUTSPRITE 3,(X,Y+16),,11
700 PUTSPRITE 2,(X,Y+8),,7
710 PUTSPRITE 3,(X,Y+16),,12
810 PUTSPRITE 1,(X,Y),,0
820 PUTSPRITE 2,(X,Y+8),,0
830 PUTSPRITE 3,(X,Y+16),,0
860 PUTSPRITE 1,(X,Y),,3
870 PUTSPRITE 2,(X,Y+8),,8
880 PUTSPRITE 3,(X,Y+16),,13
900 PUTSPRITE 2,(X,Y+8),,9
910 PUTSPRITE 3,(X,Y+16),,14
```

PROGRAMA 6.4: Para M.S.X. cambiar la línea 60 por 'A=RND(-TIME)' y en donde ponga 'RND' sustituirlo por 'RND(1)'

PROGRAMA 7.6: Para M.S.X. cambiar la línea 70 por 'A=RND(-TIME)' y ejn donde ponga 'RND' sustituirlo por 'RND(1)'

APENDICE C

Cada vez que se llame a este apéndice será para sustituir todas las sentencias 'LOCATE' por el programa que se da a continuación:

```
9000 REM **************************
9001 REM * LOCATE EN EL COMMODORE *
9002 REM **************************
9003 REM
9004 REM ***************************************
9005 REM * LA FORMA DE UTILIZARLO SERA LA SIG. *
9006 REM *                                     *
9007 REM * ASIGNAR A XP LA COLUMNA DESEADA     *
9008 REM * ASIGNAR A YP LA FILA DESEADA        *
9009 REM * HACER 'GOSUB 9000'                  *
9010 REM * AL RETORNAR LA RUTINA EL CURSOR SE  *
9011 REM * ENCONTRARA EN LA POSICION XP,YX     *
9012 REM * LISTO PARA IMPRIMIR.                *
9013 REM ***************************************
9014 REM
9020 PRINT "<HOME>";:REM ESTO SIGNIFINA PULSA LA TECLA HOME DESPUES DE LAS
    COMILLAS Y LUEGO CERRAR COMILLAS.(APARECERA UN CARACTER)
9030 FOR I=1 TO X:PRINT "->";:NEXT I:REM PULSA FLECHA HACIA LA DERECHA
9040 FOR I=1 TO Y:PRINT "  ";:NEXT I:REM PULSAR FLECHA HACIA ABAJO
9050 NEXT I
```

```
PROGRAMA 2.1: Para el COMMODORE sustituir la línea 40 por :
          40 PRINT "<SHIFT-HOME>"

C>
```

ENCICLOPEDIA PRACTICA DE LA

INFORMATICA APLICADA

INDICE GENERAL

1 COMO CONSTRUIR JUEGOS DE AVENTURA

Descripción y ejemplos de las principales familias de juegos de aventura para ordenador: simuladores de combate, aventuras espaciales, búsquedas de tesoros..., terminando con un programa que permite al lector construir sus propios libros de multiaventura.

2 COMO DIBUJAR Y HACER GRAFICOS CON EL ORDENADOR

Desde el primer «brochazo» aprenderá a diseñar y colorear tanto figuras sencillas como las más sofisticadas creaciones que pueda llegar a imaginar, sin necesidad de profundos conocimientos informáticos ni artísticos.

3 PROGRAMACION ESTRUCTURADA EN EL LENGUAJE PASCAL

Invitación a programar en PASCAL, lenguaje de alto nivel que permite programar de forma especialmente bien estructurada, tanto para aquellos que ya han probado otros lenguajes como para los que se inician en la Informática.

4 COMO ELEGIR UNA BASE DE DATOS

Libro eminentemente práctico con numerosos cuadros y tablas, útil para poder conocer las bases de datos y elegir la que más se adecúe a nuestras necesidades.

5 AÑADA PERIFERICOS A SU ORDENADOR
Breve descripción de varios periféricos que facilitan la comunicación con el ordenador personal, con algunos ejemplos de fácil construcción: ratón, lápiz óptico, marco para pantalla táctil...

6 GRAFICOS ANIMADOS CON EL ORDENADOR
En este libro las técnicas utilizadas para la animación son el resultado de unas pocas ideas básicas muy sencillas de comprender. Descubrirá los trucos y secretos de movimientos, choques, rebotes, explosiones, disparos, saltos, etc.

7 PRACTIQUE MATEMATICAS Y ESTADISTICA CON EL ORDENADOR
En este libro se repasan los principales conceptos de las Matemáticas y la Estadística, desde un punto de vista eminentemente práctico y para su aplicación al ordenador personal. Se basan los diferentes textos en la presentación de pequeños programas (que usted podrá introducir en su ordenador personal).

8 APL: LENGUAJE PARA PROGRAMADORES DIFERENTES
APL es un lenguaje muy potente que proporciona gran simplicidad en el desarrollo de programas y al mismo tiempo permite programar sin necesidad de conocer todos los elementos del lenguaje. Por ello es ideal para quienes reúnan imaginación y escasa formación en Informática.

9 DISPOSITIVOS INTERACTIVOS PARA SU ORDENADOR
Descripción detallada de la forma de construir, paso a paso y en su propia casa, dispositivos electrónicos que aumentarán la potencia y facilidad de uso de su ordenador: tableta digitalizadora, convertidores de señales analógicas, comunicaciones entre ordenadores.

10 CRIPTOGRAFIA: LA OCULTACION DE MENSAJES Y EL ORDENADOR
En este libro se presentan las técnicas de ocultacion de mensajes a través de la criptografía desde los primeros tiempos hasta la actualidad, en que el uso de los computadores ha proporcionado la herramienta necesaria para llegar al desarrollo de esta ciencia.

11 PRACTIQUE CIENCIAS NATURALES CON EL ORDENADOR
Ejemplos sencillos para practicar con el ordenador. Casos curiosos de la Naturaleza en forma de programas para su ordenador personal.

12 JUEGOS INTELIGENTES EN MICROORDENADORES
Los ordenadores pueden enfrentarse de forma «inteligente» ante puzzles y otros tipos de juegos. Esto es posible gracias al nuevo enfoque que ha dado la IA a la tradicional teoría de juegos.

13 ECONOMIA DOMESTICA CON EL ORDENADOR PERSONAL

Breve introducción a la contabilidad de doble partida y su aplicación al hogar, con explicaciones de cómo utilizar el ordenador personal para facilitar los cálculos, mediante un programa especialmente diseñado para ello.

14 COMO SIMULAR CIRCUITOS ELECTRONICOS EN EL ORDENADOR

Introducción a los diferentes métodos que se pueden emplear para simular y analizar circuitos electrónicos, mediante la utilización de diferentes lenguajes.

15 LOS LENGUAJES DE LA INTELIGENCIA ARTIFICIAL

Libro en que se describen los lenguajes específicos para la «elaboración del saber» y los entornos de programación correspondientes. El conocimiento de estos lenguajes, además de interesante en sí mismo, es sumamente útil para entender todo lo que la Informática Artificial supondrá para el futuro de la Informática.

16 PRACTIQUE FISICA Y QUIMICA CON SU ORDENADOR

Libro eminentemente práctico para realizar pequeños «experimentos» con su ordenador y distraerse de un modo útil.

17 EL ORDENADOR Y LA LITERATURA

En este libro se examinan procesadores de textos, programas de análisis literario y una curiosa aplicación desarrollada por el autor: APOLO, un programa que compone estructuras poéticas.

18 COMO ELEGIR UNA HOJA ELECTRONICA DE CALCULO

En este título se estudian las diferentes versiones existentes de esta aplicación típica, desde el punto de vista de su utilidad para, en función de las necesidades de cada usuario y del ordenador de que dispone, poder elegir aquella que más se adecúe a cada caso.

19 DIBUJOS TRIDIMENSIONALES EN EL ORDENADOR PERSONAL

Compruebe que también con su ordenador personal puede llegar a diseñar y calcular imágenes en tres dimensiones con técnicas semejantes a las utilizadas por los profesionales del dibujo con equipos mucho más sofisticados.

20 ¿MAQUINAS MAS EXPERTAS QUE LOS HOMBRES?

Después de situar los «sistemas expertos» en el contexto de la inteligencia artificial y describir su construcción, su funcionamiento, su utilidad, etc., se analiza el papel que pueden tener en el futuro (y presente, ya) de la Informática.

21 PRACTIQUE HISTORIA Y GEOGRAFIA CON SU ORDENADOR
Libro interesante para los aficionados a estas ciencias, a quienes presenta una nueva visión de cómo utilizar el microordenador en su estudio.

22 ERGONOMIA: COMUNICACION EFICIENTE HOMBRE-MAQUINA
Análisis de la comunicación entre el hombre y la máquina, y estudio de diferentes soluciones que tienden a facilitarla lo más posible.

23 EL ORDENADOR Y LA ASTRONOMIA
Los cálculos astronómicos y el conocimiento del firmamento en un libro apasionante y curioso.

24 VISION ARTIFICIAL. TRATAMIENTO DE IMAGENES POR ORDENADOR
El procesado de imágenes es un campo de reciente y rápido desarrollo con importantes aplicaciones en áreas tan diversas como la mejora de imágenes biomédicas, robóticas, teledetección y otras aplicaciones industriales y militares. Se presentan los principios básicos, los sistemas y las técnicas de procesado más usuales.

25 LA ESTACION TERMINAL PERSONAL
Las modernas técnicas de comunicación van permitiendo que las grandes capacidades de proceso y el acceso a bases de datos de gran tamaño estén cada día más al alcance de cada usuario (fuera ya de los Centros de Proceso de Datos).

26 EL ORDENADOR COMO MAQUINA DE ESCRIBIR INTELIGENTE
Descripción de los sistemas de tratamiento de textos existentes, análisis comparativos y estudio de posibilidades de cada uno de ellos. Guía práctica para la elección del presente paquete que más se adecúe a nuestras necesidades y al ordenador personal de que dispongamos.

27 EL LENGUAJE C, PROXIMO A LA MAQUINA
Lenguaje de programación que se está imponiendo en los microordenadores más grandes, tanto por su facilidad de aprendizaje y uso, como por su enorme potencia y su adecuación a la programación estructurada. Vinculado íntimamente al sistema operativo UNIX es uno de los lenguajes de más futuro entre los que utilizan los micros personales.

28 EL ORDENADOR COMO INSTRUMENTO MUSICAL Y DE COMPOSICION

Análisis de cómo se puede utilizar el ordenador para la composición o interpretación de música. Libro eminentemente práctico, con numerosos ejemplos (que usted podrá practicar en su ordenador casero) y lleno de sugerencias para disfrutar haciendo de su ordenador un verdadero instrumento musical.

29 LA CREATIVIDAD EN EL ORDENADOR. EXPERIENCIAS EN LOGO

El LOGO es un lenguaje enormemente capacitado para la creación principalmente gráfica y en especial para los niños. En este sentido se han desarrollado numerosas experiencias. En el libro se analizan estas experiencias y las posibilidades del LOGO en este sentido, así como su aplicación a su ordenador casero para que usted mismo (o con sus hijos) pueda repetirlas.

30 SISTEMAS OPERATIVOS: EL SISTEMA NERVIOSO DEL ORDENADOR

Características de diversos sistemas operativos utilizados en los ordenadores personales y caseros. Se trata de llegar al conocimiento, ameno, aunque riguroso, de la misión del sistema operativo de su ordenador, para que usted consiga sacar mayor rendimiento a su equipo.

NOTA: Ediciones Siglo Cultural, S. A., se reserva el derecho de modificar, sin previo aviso, el orden, titulo o contenido de cualquier volumen de la colección.

Al hablar de ordenadores, generalmente se piensa en el divertido mundo de los videojuegos. En ocasiones se habrá preguntado cómo es posible el movimiento en un ordenador.

En este libro las técnicas utilizadas para la animación son el resultado de unas pocas ideas básicas muy sencillas de comprender.

Descubrirá los trucos y secretos de movimientos, choques, rebotes, explosiones, disparos, saltos, etc..., es decir, los componentes básicos para llegar a entender el mundo en movimiento.

Encontrará que tanto el más sencillo programa de desplazamiento de una «O» como el aparentemente complejo movimiento de un personaje se basan en los mismos principios básicos.